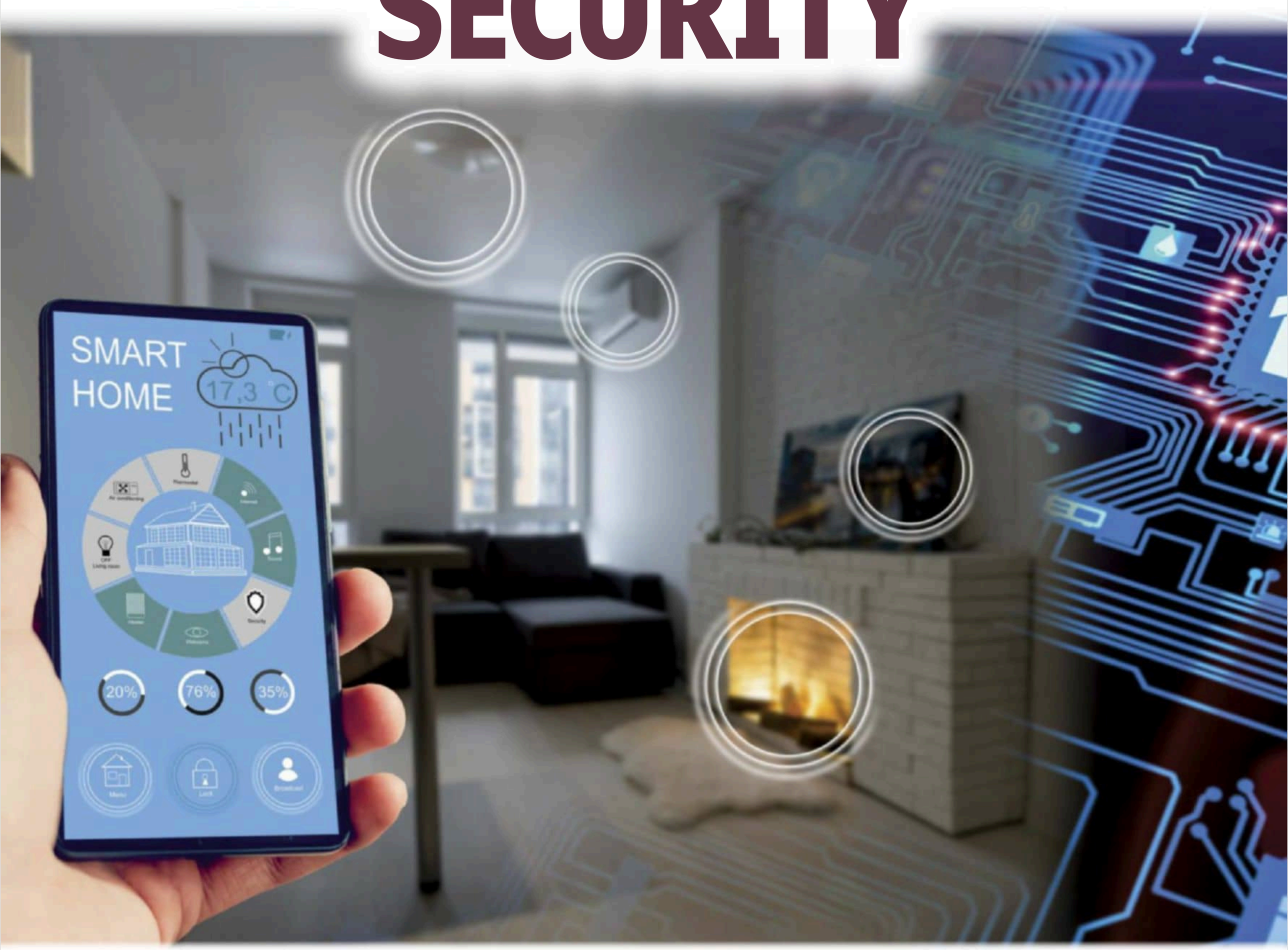
CONSUMER ELECTRONICS
JANUARY 2025
ISSUE 414

circuitcellar.com

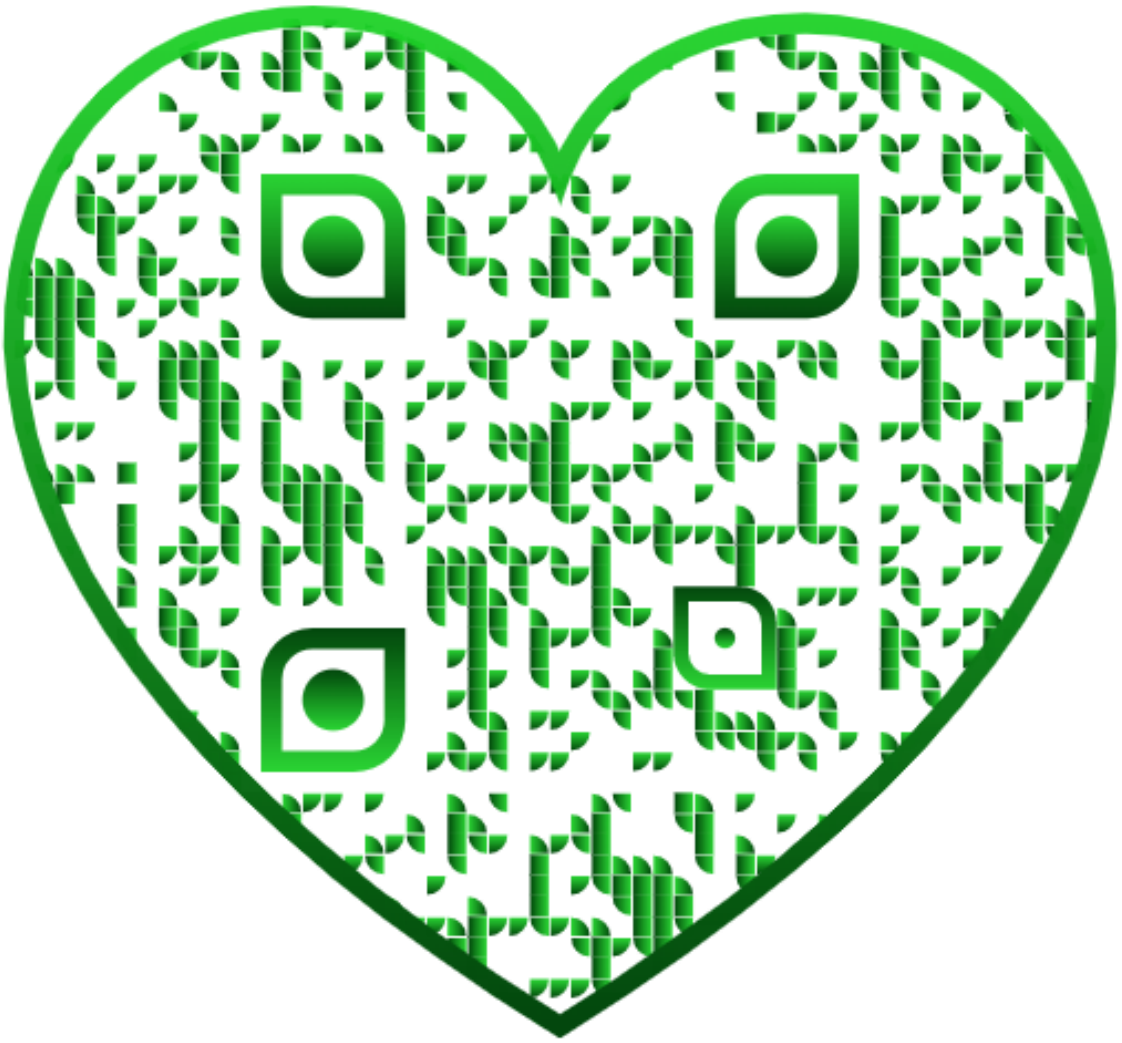
circuit cellar
Inspiring the Evolution of Embedded Design

SMART HOME SECURITY



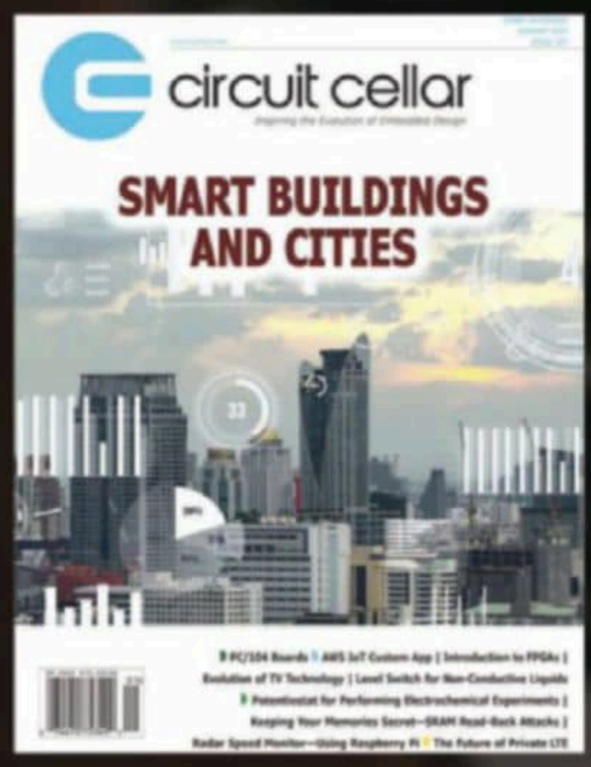
$9.00US $10.00CAN
01>
0 74470 75349 0

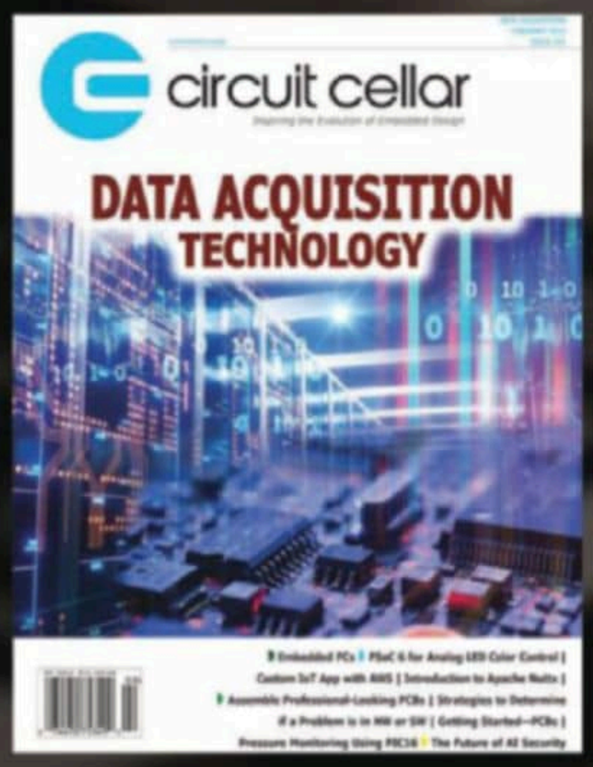
- Low Power MCUs
- Solutions for Performance Bottlenecks in Embedded Linux | PCBs with a CNC Milling Machine | Wi-Fi Router Watchdog With Raspberry Pi Pico | Basic Pulse Circuits Flip-Flops
- Raspberry Pi Pico to Hack an IP Camera | Amazon Alexa to Control Custom IoT Gadgets
- The Future of IoT Sustainability

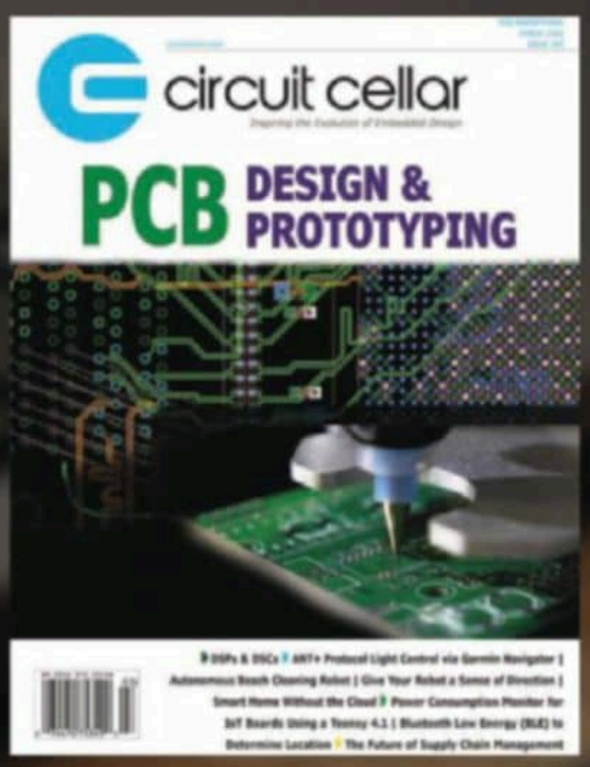


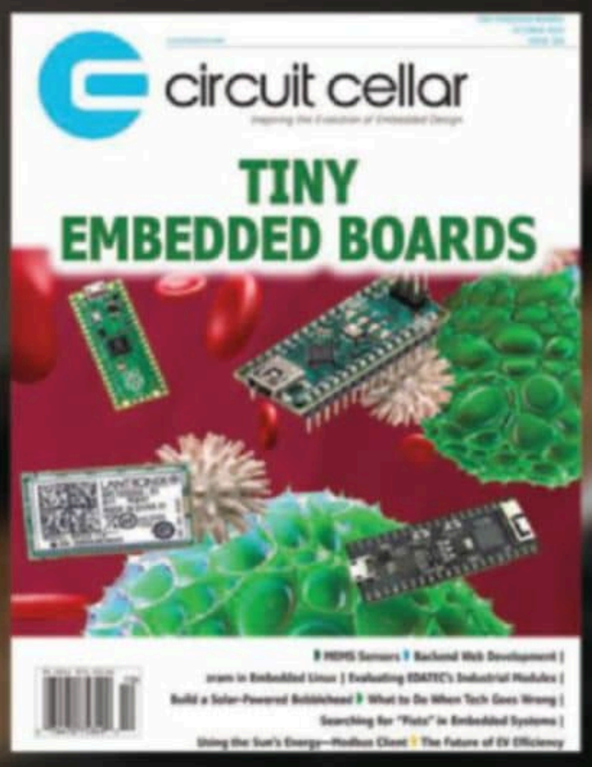
Imagine what you will Discover

CC VAULT





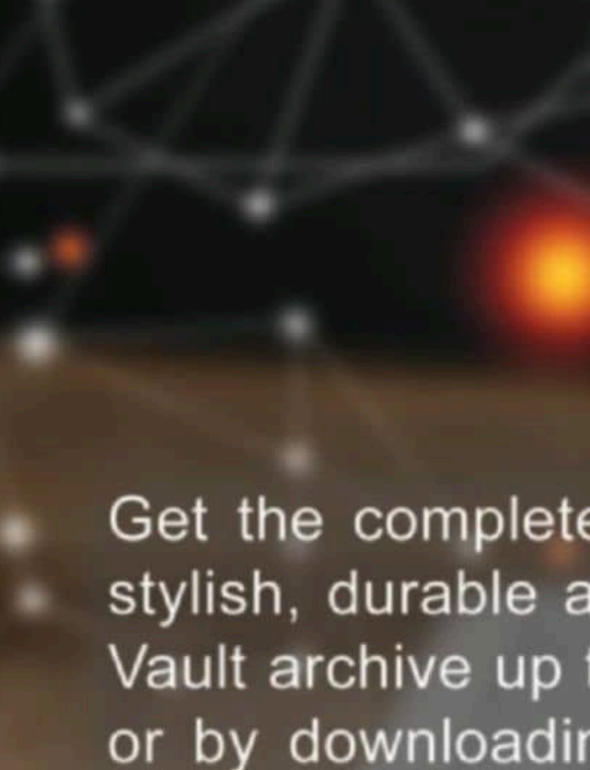


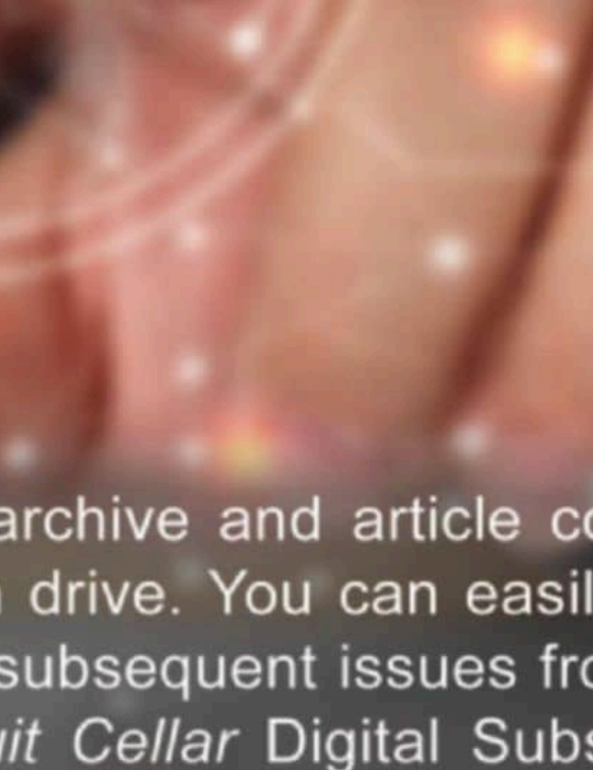


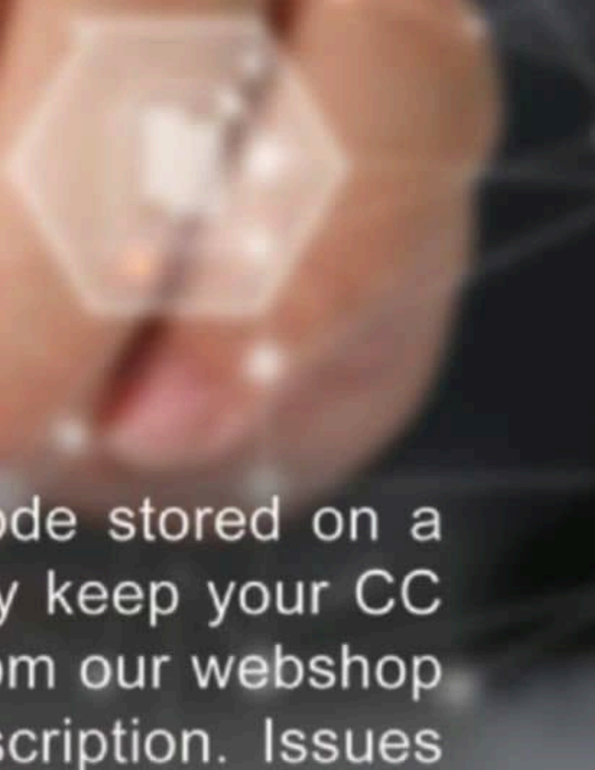
400+ ISSUES

Get the complete *Circuit Cellar* issue archive and article code stored on a stylish, durable and portable USB flash drive. You can easily keep your CC Vault archive up to date by purchasing subsequent issues from our webshop or by downloading issues with a *Circuit Cellar* Digital Subscription. Issues appear in searchable PDF format.

cc-webshop.com







Complete archive includes PDFs of all issues in print through date of purchase.

INPUT Voltage

Sustainability and Security

As a new year dawns, it's time to take inventory of all the priorities that we identified last year and determine whether we fulfilled the promises we made to ourselves. I mention this because this month's feature, "Smart Home Lock Down," examines the latest efforts to instill sustainability and security in the embedded design world.

Although the theme of this month's feature topic was consumer electronics, I found it to be an extremely broad category. Searching for hot trends here provides some difficulty in narrowing down the choices. But finding a thread to guide the story became easier once the topics of sustainability and security surfaced. In this feature, you will find the latest efforts by Powercast to make IoT systems more sustainable by reducing battery dependence. And then the story turns to Matter, the wireless communication standard that enables developers to add security to IoT devices, according to Steve Hanna, a Distinguished Engineer with Infineon Technologies.

Have you, as a developer, given the appropriate priority to security and sustainability in system design? Is it worth revisiting the promises made in 2024 to determine if goals were achieved, or is more focus on the subject required in 2025? Perhaps your priorities to improve system design lie elsewhere, but sustainability and security are given their due here to kick off the new year.

Perhaps system performance will be a design objective in 2025. In that case, returning writer Pedro Bertoleti provides an update on Linux performance bottlenecks and how to eliminate them. After all, performance is the key metric by which end users evaluate a new system. It's best to leave a very good first impression.

If ultra-low power is one of the leading priorities, then the datasheet on this issue examines nine MCUs from industry vendors. You will see selections from Analog Devices, Infineon, Microchip, NXP, Silicon Labs, STMicroelectronics, and others to deliver the lowest microamps per megahertz.

Whenever a new design calls for a unique enclosure or distinctively shaped printed circuit board, finding a pre-made selection is not always easy or inexpensive. Matt Oppenheim tells us how to create a custom PCB using a CNC milling machine and KiCad, CopperCAM, and Candle software.

Then, there are those pesky circuits that require input from the real world. Wolfgang Matthes offers the second part of a three-part article series on Basic Pulse Circuits. He dives into enabling flip-flops, timing parameters and synchronization, and design tasks needed to capture, detect, and filter pulses.

The beginning of a new year could also be perfect for fine-tuning hardware hacking skills. Colin Flynn shows us how a low-cost Raspberry Pi Pico and a TP-Link Tapo C200 smart IP camera could become the next automated bird deterrent or a home automation server.

Also, all those custom IoT gadgets built and installed over the years have served you well in your home automation scheme, using a PC or smartphone as the interface. But can you control them with voice commands over Amazon Alexa? Brian Millier shows us how to integrate IoT gadgets with Amazon Echo by using emulation to receive spoken alarms.

Tom Murphy
Editor-In-Chief
tmurphy@circuitcellar.com

Issue 414 January 2025 | ISSN 1528-0608

CIRCUIT CELLAR® (ISSN 1528-0608) is published monthly by:

KCK Media Corp.
PO Box 417, Chase City, VA 23924

Periodical rates paid at Chase City, VA, and additional offices. One-year (12 issues) subscription rate US and possessions $50, Canada $65, Foreign/ ROW $75. All subscription orders payable in US funds only via Visa, MasterCard, international postal money order, or check drawn on US bank.

SUBSCRIPTION MANAGEMENT

Online Account Management: circuitcellar.com/account
Renew | Change Address/E-mail | Check Status

CUSTOMER SERVICE

E-mail: customerservice@circuitcellar.com
Phone: 434.533.0246
Mail: Circuit Cellar, PO Box 417, Chase City, VA 23924

Postmaster: Send address changes to
Circuit Cellar, PO Box 417, Chase City, VA 23924

NEW SUBSCRIPTIONS

circuitcellar.com/subscription

ADVERTISING

Contact: Hugh Heinsohn
Phone: 757-525-3677
Fax: 888-980-1303
E-mail: hheinsohn@circuitcellar.com
Advertising rates and terms available on request.

NEW PRODUCTS

E-mail: product-editor@circuitcellar.com

HEAD OFFICE

KCK Media Corp.
PO Box 417
Chase City, VA 23924
Phone: 434-533-0246

COPYRIGHT NOTICE

Entire contents copyright © 2024 by KCK Media Corp. All rights reserved. Circuit Cellar is a registered trademark of KCK Media Corp. Reproduction of this publication in whole or in part without written consent from KCK Media Corp. is prohibited.

DISCLAIMER

KCK Media Corp. makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors printed in Circuit Cellar®. Furthermore, because of possible variation in the quality and condition of materials and workmanship of reader-assembled projects, KCK Media Corp. disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published in Circuit Cellar®.

The information provided in Circuit Cellar® by KCK Media Corp. is for educational purposes. KCK Media Corp. makes no claims or warrants that readers have a right to build things based upon these ideas under patent or other relevant intellectual property law in their jurisdiction, or that readers have a right to construct or operate any of the devices described herein under the relevant patent or other intellectual property law of the reader's jurisdiction. The reader assumes any risk of infringement liability for constructing or operating such devices.

© KCK Media Corp. 2024 Printed in the United States

OUR NETWORK

circuit cellar
LinuxGizmos
audioxpress
VOICE COIL
LIS LOUDSPEAKER INDUSTRY SOURCEBOOK

SUPPORTING COMPANIES

Black Magic	11
BusBoard Prototype Systems	15
CCS, Inc.	61
DesignCon 2025	21
eCon Systems	60
EmbeddedTS	61, C4
Embedded World 2025	35
Microchip	17
Myir	19
Saelig	55
SeaFire Micros, Inc.	C3

NOT A SUPPORTING COMPANY YET?

Contact Hugh Heinsohn
hugh@circuitcellar.com,
Phone: 757-525-3677,
Fax: 888-980-1303
to reserve space in the
next issue of *Circuit Cellar.*

THE TEAM

FOUNDER Steve Ciarcia
PUBLISHER KC Prescott
CONTROLLER Chuck Fellows
EDITOR-IN-CHIEF Tom Murphy
CONTRIBUTING EDITOR Brian Millier
CONTENT MANAGER- EU Jurij Mikeln
SENIOR ASSOCIATE EDITOR Shannon Becker
TECHNICAL COPY EDITOR Carol Bower
PROJECT EDITORS Ken Davidson, David Tweed
MARKETING MANAGER Tori Zienka
ADVERTISING COORDINATOR Diana Winsor
ADVERTISING SALES REP. Hugh Heinsohn

COLUMNISTS
Stuart Ball (Start to Finish)
Bob Japenga (Embedded in Thin Slices)
Brian Millier (Picking Up Mixed Signals)
Colin O'Flynn (Embedded Systems Essentials)

FEATURES

4 **Performance Bottlenecks in Embedded Linux Solutions**
Analysis, Identification, and Mitigation
By Pedro Bertoleti

12 **Create Your Own PCBs with a CNC Milling Machine**
Using KiCad, CopperCAM, and Candle Software
By Matthew Oppenheim

22 **Building a Wi-Fi Router Watchdog**
With Raspberry Pi Pico, MicroPython, and Thonny IDE
By Dev Gualtieri

26 **Basic Pulse Circuits**
Part 2: Flip-Flops, Timing Parameters and Synchronization
By Wolfgang Matthes

36 TECHNOLOGY FEATURE
Smart Home Lock Down
Matter Provides Security Blanket
By Tom Murphy

42 DATASHEET
Microamps Per Megahertz
Ultra-Low Power MCUs Minimize Current Consumption
By Tom Murphy

COLUMNS

46 Embedded System Essentials
Holiday Hangover Hardware Hacking
Using a Raspberry Pi Pico to Hack an IP Camera
By Colin O'Flynn

50 Picking Up Mixed Signals
Using Amazon Alexa to Control Custom IoT Gadgets
Part 2: Emulation and Spoken Alarms
By Brian Millier

62 TECH THE FUTURE
The Future of Future of IoT Sustainability
Atmosic Solving Key IoT Challenges
Cutting Battery Usage by Advancing Sustainability and Interoperability
By Paul Davis

57 : PRODUCT NEWS
60 : TEST YOUR EQ

@editor_cc
@circuitcellar
circuitcellar
circuit_cellar

FEATURES

Performance Bottlenecks in Embedded Linux Solutions

Analysis, Identification, and Mitigation

By
Pedro Bertoleti

Good performance is a requirement for every technology, and system designers rely on operating systems to ensure fast and smooth transitions in critical applications. Fortunately, Pedro writes, the embedded Linux OS offers ways for finding, analyzing and mitigating performance bottlenecks so embedded systems can deliver the speed and efficiency that end users expect.

Performance bottlenecks in embedded systems are limitations that restrict a system's speed and efficiency. This article identifies the common performance bottlenecks in embedded Linux solutions, and shows us some methods and tools for identifying, analyzing, and mitigating them.

The world of technology today seems to obey the famous quote of philosopher Heraclitus of Ephesus, who said, "The only constant is change." In terms of innovation, technology is one of the most dynamic: Rapidly evolving, and changes can seemingly occur overnight. There's one exception, however: Good performance. This is a constant requirement for every single technology. Technology exhibiting poor performance won't be used extensively, end users won't like it, and it will quickly die.

For embedded operating systems (OS), good performance is critical because the software often handles critical solutions that must be fast and smooth, like safety controls in motor vehicles. This is also true for embedded Linux.

This article explores some methods and tools for analyzing the performance bottlenecks of embedded Linux—helping to identify them and giving some insights on how to mitigate them.

PERFORMANCE BOTTLENECKS

Unfortunately, the term, "performance bottlenecks analysis" isn't very precise, as long as it can comprise a wide range of definitions, from user experience metrics to specific hardware metrics. This means that first, you must determine a common base for the analysis, which corresponds to defining which topics (or points of view) will be taken in consideration.

The common base for performance bottlenecks analysis focuses on the three of the most important resources in any embedded system: CPU (central processing unit), random access memory (RAM), and swap memory (the part of a computer's hard drive where data is stored temporarily, when the RAM is full). To better detail this analysis, the following topics will be covered:

- CPU usage analysis, for both specific processes and general system usage
- RAM usage analysis, for both specific processes and general system usage
- Swap memory usage analysis
- Memory thrashing analysis
- Memory leak analysis

CPU USAGE ANALYSIS

Proper CPU usage in embedded systems running Linux—in fact, for any embedded

systems solution—is crucial for maintaining overall system performance or optimizing power consumption. Excessive or prolonged high CPU usage not only increases power consumption and raises the temperature of the SoC (system on chip) or SiP (system in package) but can also cause thermal throttling. Thermal throttling leads to general embedded Linux slow-down, significantly affecting its efficiency and user experience, as shown in **Figure 1**.

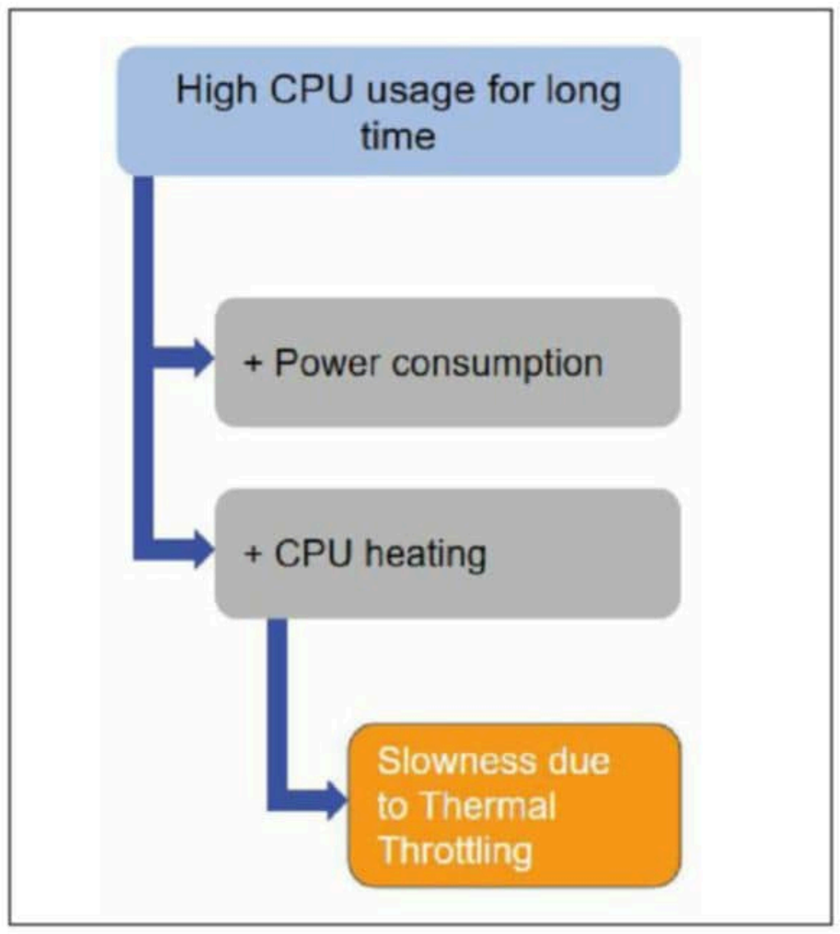


FIGURE 1
Thermal throttling leads to general embedded Linux slow-down, significantly affecting efficiency and user experience.

When an embedded Linux solution shows signs of sluggishness and bad performance, one of the first things to do is to check if the CPU usage (both per-process and overall cases) is consistently high over time. If it is, then one of two scenarios is likely to be true:

The SoC or SiP being used has CPUs that are undersized for the intended solution. It means the SoC or SiP cannot deliver the needed processing power.

The SoC or SiP has sufficient processing power for the intended solution, but due to development issues, bad programming practices, and/or errors, certain processes are demanding far more CPU resources than expected.

In the first case, there's no easy way out; the solution is a hardware upgrade to a more capable SoC or SiP, with more powerful CPUs, since the current hardware lacks the required processing power. In the second scenario, however, optimizing software design or addressing any bugs to the process source-code responsible group, enterprise, or developer can significantly improve CPU usage, resulting in better overall performance.

But how can CPU usage, both overall and by individual processes, be measured to determine if optimization is possible? Several measurement tools are available on Linux, one of the most common of which is the "htop."

The htop is an interactive system monitor, process viewer, and manager, tailored for Unix-based systems, which is the case of Linux, a derivation from Minix, which is based

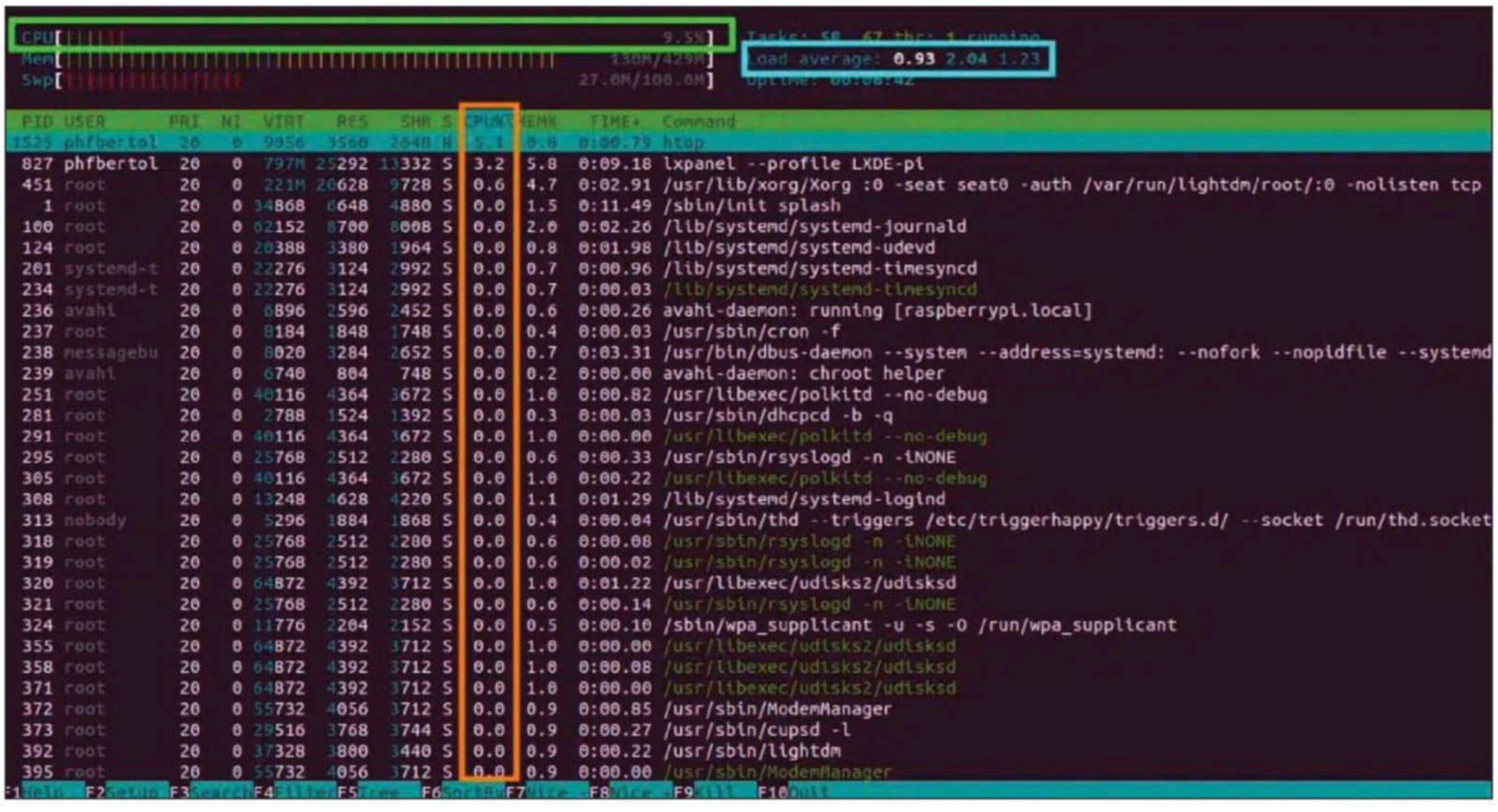

CPU[9.5%]
Mem[130M/429M]
Swp[27.0M/100.0M]
Load average: 0.93 2.04 1.23
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
1525 phfbertol 20 0 9056 3560 2648 R 5.1 0.8 0:00.79 htop
827 phfbertol 20 0 797M 25292 13332 S 3.2 5.8 0:09.18 lxpanel --profile LXDE-pi
451 root 20 0 221M 20628 9728 S 0.6 4.7 0:02.91 /usr/lib/xorg/Xorg :0 -seat seat0 -auth /var/run/lightdm/root/:0 -nolisten tcp
1 root 20 0 34868 6648 4880 S 0.0 1.5 0:11.49 /sbin/init splash
100 root 20 0 62152 8700 8008 S 0.0 2.0 0:02.26 /lib/systemd/systemd-journald
124 root 20 0 20388 3380 1964 S 0.0 0.8 0:01.98 /lib/systemd/systemd-udevd
201 systemd-t 20 0 22276 3124 2992 S 0.0 0.7 0:00.96 /lib/systemd/systemd-timesyncd
234 systemd-t 20 0 22276 3124 2992 S 0.0 0.7 0:00.03 /lib/systemd/systemd-timesyncd
236 avahi 20 0 6896 2596 2452 S 0.0 0.6 0:00.26 avahi-daemon: running [raspberrypi.local]
237 root 20 0 8184 1848 1748 S 0.0 0.4 0:00.03 /usr/sbin/cron -f
238 messagebu 20 0 8020 3284 2652 S 0.0 0.7 0:03.31 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd
239 avahi 20 0 6740 804 748 S 0.0 0.2 0:00.00 avahi-daemon: chroot helper
251 root 20 0 40116 4364 3672 S 0.0 1.0 0:00.82 /usr/libexec/polkitd --no-debug
281 root 20 0 2788 1524 1392 S 0.0 0.3 0:00.03 /usr/sbin/dhcpcd -b -q
291 root 20 0 40116 4364 3672 S 0.0 1.0 0:00.00 /usr/libexec/polkitd --no-debug
295 root 20 0 25768 2512 2280 S 0.0 0.6 0:00.33 /usr/sbin/rsyslogd -n -iNONE
305 root 20 0 40116 4364 3672 S 0.0 1.0 0:00.22 /usr/libexec/polkitd --no-debug
308 root 20 0 13248 4628 4220 S 0.0 1.1 0:01.29 /lib/systemd/systemd-logind
313 nobody 20 0 5296 1884 1868 S 0.0 0.4 0:00.04 /usr/sbin/thd --triggers /etc/triggerhappy/triggers.d/ --socket /run/thd.socket
318 root 20 0 25768 2512 2280 S 0.0 0.6 0:00.08 /usr/sbin/rsyslogd -n -iNONE
319 root 20 0 25768 2512 2280 S 0.0 0.6 0:00.02 /usr/sbin/rsyslogd -n -iNONE
320 root 20 0 64872 4392 3712 S 0.0 1.0 0:01.22 /usr/libexec/udisks2/udisksd
321 root 20 0 25768 2512 2280 S 0.0 0.6 0:00.14 /usr/sbin/rsyslogd -n -iNONE
324 root 20 0 11776 2204 2152 S 0.0 0.5 0:00.10 /sbin/wpa_supplicant -u -s -O /run/wpa_supplicant
355 root 20 0 64872 4392 3712 S 0.0 1.0 0:00.00 /usr/libexec/udisks2/udisksd
358 root 20 0 64872 4392 3712 S 0.0 1.0 0:00.08 /usr/libexec/udisks2/udisksd
371 root 20 0 64872 4392 3712 S 0.0 1.0 0:00.00 /usr/libexec/udisks2/udisksd
372 root 20 0 55732 4056 3712 S 0.0 0.9 0:00.85 /usr/sbin/ModemManager
373 root 20 0 29516 3768 3744 S 0.0 0.9 0:00.27 /usr/sbin/cupsd -l
392 root 20 0 37328 3800 3440 S 0.0 0.9 0:00.22 /usr/sbin/lightdm
395 root 20 0 55732 4056 3712 S 0.0 0.9 0:00.00 /usr/sbin/ModemManager
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice -F8Nice +F9Kill F10Quit

FIGURE 2
Screenshot of the htop tool, showing the visualization of three aspects of CPU usage.

on Unix. Built as an alternative to the Linux utility called "top" (Table of Processes), the htop provides similar functionality but with enhanced flexibility, a nicer appearance, and customization options for viewing and managing system processes. The htop tool provides a real-time, dynamic view of various Linux system metrics, including CPU usage, RAM usage, swap usage, process uptime, virtual memory per process.

Specifically for CPU usage, the htop allows visualization of the following features:

- Instantaneous individual CPU usage percentage for each available CPU, (highlighted in green in **Figure 2**).
- CPU usage per process, (shown in orange in Figure 2).
- Average overall CPU load over time, known as "Load Average," (highlighted in light blue in Figure 2). This includes average CPU usage over the last minute (first Load Average value), the last 5 minutes (second Load Average value), and the last 15 minutes (third Load Average value).

When using the keyboard keys from F3 to F9 in the htop tool, it's possible to search for specific processes (which is especially useful for tracking just one process among the many an embedded Linux OS system may have), filter the process view, and kill specific processes.

The htop tool is therefore highly useful for monitoring CPU usage on embedded Linux OS, providing insights into both instantaneous CPU usage (overall and per process) and the average CPU usage over time.

Another great tool for analyzing CPU usage per process is called "ps" (Process Status). The ps tool's primary function is general process monitoring, displaying metrics such as CPU usage percentage per process. Additionally, it allows the developer to check which processes are actively running, giving a comprehensive view of processes execution on an embedded Linux OS at any moment. An example output of ps tool is shown in **Figure 3**.

A third useful tool for analyzing CPU usage on embedded Linux is called "sar." The sar (System Activity Reporter) monitors overall activity of each CPU present on the solution SoC or SiP, providing metrics such as percentage of CPU usage, CPU idle time, CPU IOWait, and much more. This tool is an excellent choice for assessing CPU load and

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  1.4  1.5  34868  6648 ?        Ss   17:50   0:11 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    17:50   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   17:50   0:00 [slub_flushwq]
root           4  0.0  0.0      0     0 ?        I<   17:50   0:00 [netns]
root           8  0.0  0.0      0     0 ?        I<   17:50   0:00 [mm_percpu_wq]
root           9  0.0  0.0      0     0 ?        S    17:50   0:00 [rcu_tasks_rude_]
root          10  0.0  0.0      0     0 ?        S    17:50   0:00 [rcu_tasks_trace]
root          11  0.2  0.0      0     0 ?        S    17:50   0:02 [ksoftirqd/0]
root          12  0.0  0.0      0     0 ?        S    17:50   0:00 [kdevtmpfs]
root          13  0.0  0.0      0     0 ?        I<   17:50   0:00 [inet_frag_wq]
root          15  0.0  0.0      0     0 ?        S    17:50   0:00 [kauditd]
root          16  0.0  0.0      0     0 ?        S    17:50   0:00 [khungtaskd]
root          17  0.0  0.0      0     0 ?        S    17:50   0:00 [oom_reaper]
root          18  0.0  0.0      0     0 ?        I<   17:50   0:00 [writeback]
root          19  0.2  0.0      0     0 ?        S    17:50   0:02 [kcompactd0]
root          37  0.0  0.0      0     0 ?        I<   17:50   0:00 [kblockd]
root          38  0.0  0.0      0     0 ?        I<   17:50   0:00 [blkcg_punt_bio]
root          39  0.0  0.0      0     0 ?        S    17:50   0:00 [watchdogd]
root          41  0.1  0.0      0     0 ?        I    17:50   0:01 [kworker/u2:2-events_unbound]
root          42  0.0  0.0      0     0 ?        I<   17:50   0:00 [rpciod]
root          43  0.2  0.0      0     0 ?        I<   17:50   0:01 [kworker/u3:0-brcmf_wq/mmc1:0001:1]
root          44  0.0  0.0      0     0 ?        I<   17:50   0:00 [xprtiod]
root          45  0.2  0.0      0     0 ?        S    17:50   0:01 [kswapd0]
root          46  0.0  0.0      0     0 ?        I<   17:50   0:00 [nfsiod]
root          47  0.0  0.0      0     0 ?        I<   17:50   0:00 [iscsi_eh]
root          48  0.0  0.0      0     0 ?        I<   17:50   0:00 [iscsi_conn_clea]
root          49  0.0  0.0      0     0 ?        I<   17:50   0:00 [dwc_otg]
root          50  0.0  0.0      0     0 ?        I<   17:50   0:00 [DWC Notificatio]
root          52  0.0  0.0      0     0 ?        S<   17:50   0:00 [vchiq-slot/0]
root          53  0.0  0.0      0     0 ?        S<   17:50   0:00 [vchiq-recy/0]
root          54  0.0  0.0      0     0 ?        S<   17:50   0:00 [vchiq-sync/0]
root          55  0.0  0.0      0     0 ?        I<   17:50   0:00 [zswap-shrink]
root          57  0.0  0.0      0     0 ?        I<   17:50   0:00 [mmc_complete]
root          58  0.2  0.0      0     0 ?        I<   17:50   0:02 [kworker/0:1H-mmc_complete]
root          60  0.0  0.0      0     0 ?        S    17:50   0:00 [jbd2/mmcblk0p2-]
root          61  0.0  0.0      0     0 ?        I<   17:50   0:00 [ext4-rsv-conver]
root          62  0.0  0.0      0     0 ?        I<   17:50   0:00 [mld]
root          63  0.0  0.0      0     0 ?        I<   17:50   0:00 [ipv6_addrconf]
root         100  0.2  1.9  62152  8700 ?        Ss   17:50   0:02 /lib/systemd/systemd-journald
root         124  0.2  0.7  20388  3380 ?        Ss   17:50   0:01 /lib/systemd/systemd-udevd
root         141  0.0  0.0      0     0 ?        S    17:50   0:00 [vchiq-keep/0]
root         144  0.0  0.0      0     0 ?        S<   17:50   0:00 [SMIO]
root         170  0.0  0.0      0     0 ?        I<   17:50   0:00 [mmal-vchiq]
root         171  0.0  0.0      0     0 ?        I<   17:50   0:00 [mmal-vchiq]
root         173  0.0  0.0      0     0 ?        I<   17:50   0:00 [mmal-vchiq]
root         175  0.0  0.0      0     0 ?        I<   17:50   0:00 [mmal-vchiq]
root         176  0.0  0.0      0     0 ?        I<   17:50   0:00 [mmal-vchiq]
root         177  0.0  0.0      0     0 ?        I<   17:50   0:00 [mmal-vchiq]
root         178  0.0  0.0      0     0 ?        I<   17:50   0:00 [mmal-vchiq]
systemd+     201  0.1  0.7  22276  3124 ?        Ssl  17:50   0:00 /lib/systemd/systemd-timesyncd
avahi        236  0.0  0.5   6896  2596 ?        Ss   17:50   0:00 avahi-daemon: running [raspberrypi.local]
root         237  0.0  0.4   8184  1848 ?        Ss   17:50   0:00 /usr/sbin/cron -f
message+     238  0.4  0.7   8020  3284 ?        Ss   17:50   0:03 /usr/bin/dbus-daemon --system --address=systemd: --no
avahi        239  0.0  0.1   6740   804 ?        S    17:50   0:00 avahi-daemon: chroot helper
```

FIGURE 3
Example of the ps tool's output.

obtaining a comprehensive view of CPU usage by the embedded Linux system.

It's possible to configure the sar tool, for setting the monitoring interval (time between CPU features measurements) and specifying which CPUs the tool will monitor. For example, to monitor the first CPU available on a given SoC or SiP, with a 500ms interval between CPU features readings, the following command can be used:

```
sar -p 1 500
```

From a practical standpoint, here are some strategies for investigating CPU usage in embedded Linux OS:

Step 1: Check the Load Average (over the last 1, 5, and 15 minutes) to see if the CPUs are undersized for the intended solution. If so, proceed to Step 2.

Step 2: Use the individual CPU usage percentage to check if any process or application is consuming more CPU than expected (or maintaining a consistently high usage). If none are found, the overall set of software in the solution may exceed what the SoC or SiP CPUs can support. In this case, consider optimizing for IOWait to help minimize performance issues. If a process with anomaly high CPU usage is found, move to Step 3.

Step 3: If the process using excessive CPU is one you developed, investigate possible optimizations regarding too many threads being used, waste CPU time on unnecessary code blocks, and so on. If it is third-party software, look for updates, or even consider replacing it with a more efficient alternative.

Another good approach is to rewrite processes (obviously, processes you or your teams are responsible for) in a more efficient programming language. In many cases, processes developed using higher-level languages require more runtime and processing, leading to increased CPU usage over time. It could be a good idea to assess whether, for certain CPU-intensive processes and applications, using a language with a shorter runtime might be beneficial. See **Table 1** where a programming language benchmark [1] is shown, highlighting that the Python programming language can be almost 76 times slower than C.

RAM USAGE ANALYSIS

Efficient RAM usage is essential for optimal performance in embedded software, because during system active use, RAM stores all applications, processes, and essential elements needed for the operating system (in this case, embedded Linux) to run and use—with access times far faster than mass storage memory (hard disks, for instance).

Excessive RAM usage by certain processes can lead to a situation where the available (or free) RAM is insufficient to handle all processes, temporary data (buffers), and other essentials needed for the embedded Linux OS to operate smoothly. This forces the OS to manage RAM through memory swapping or even force processes termination to free memory, leading to a significant performance reduction. In extreme cases, this can result in "memory thrashing," where the time spent managing memory paging becomes equally or more significant than the time spent running necessary processes, themselves, causing severe system sluggishness. The topics of memory swap and memory thrashing will be discussed in detail later in this article.

When an embedded Linux-based solution exhibits bad performance, alongside checking CPU usage, one of the steps should be to assess whether RAM usage by the solution's processes is excessive (more than expected, considering an analysis of RAM consumption over time). If so, this indicates a RAM usage issue, which could stem from excessive RAM consumption by certain processes due to

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Table 1
This programming language benchmark highlights that Python can be almost 76 times slower than C. (Source: The New Stack website [1].)

software design issues, or even a memory leak (a condition that will also be covered in detail later in this article).

All Linux tools discussed so far—the htop, ps, and sar tools—can also be used to monitor RAM usage. In the htop tool, each reported process includes a percentage of memory used (labeled MEM%), as well as total memory in use and swap usage. Thus, the htop tool is highly useful for investigating the instantaneous memory usage of one or more processes on embedded Linux OS. In **Figure 4,** it's possible to see the RAM usage percentage per process (highlighted in green) and the overall RAM and swap usage (highlighted in blue).

In addition to the htop tool, another great tool for analyzing RAM usage on embedded Linux OS is called "free." What sets the free tool apart from other tools is its ability to provide an overall view of memory usage in embedded Linux OS, including total, free, and used RAM, as well as memory shared across processes in Embedded Linux OS. Through its metrics, it's even possible to identify signs of memory leaks and memory thrashing—both of which are discussed a little later.

The following are strategies to help mitigate high RAM consumption in embedded Linux OS:

- Use ZRAM. ZRAM acts as a RAM drive, where all data is stored in a compressed format and is automatically decompressed when read. Since RAM access times are significantly faster than disk access, using ZRAM as memory swap can help minimize the slowdown caused by swap operations.
- Adjust the swappiness value (typically in /proc/sys/vm/swappiness file). This parameter controls how aggressively Linux performs swapping, allowing you to minimize swap operations within a given timeframe.
- Identify possible memory leaks. Memory leaks lead to high RAM usage and thus excessive swap and memory thrashing. If a process you or your team has developed shows increasing RAM consumption over time, you can use tools such as "valgrind" to locate memory leaks and address them accordingly.

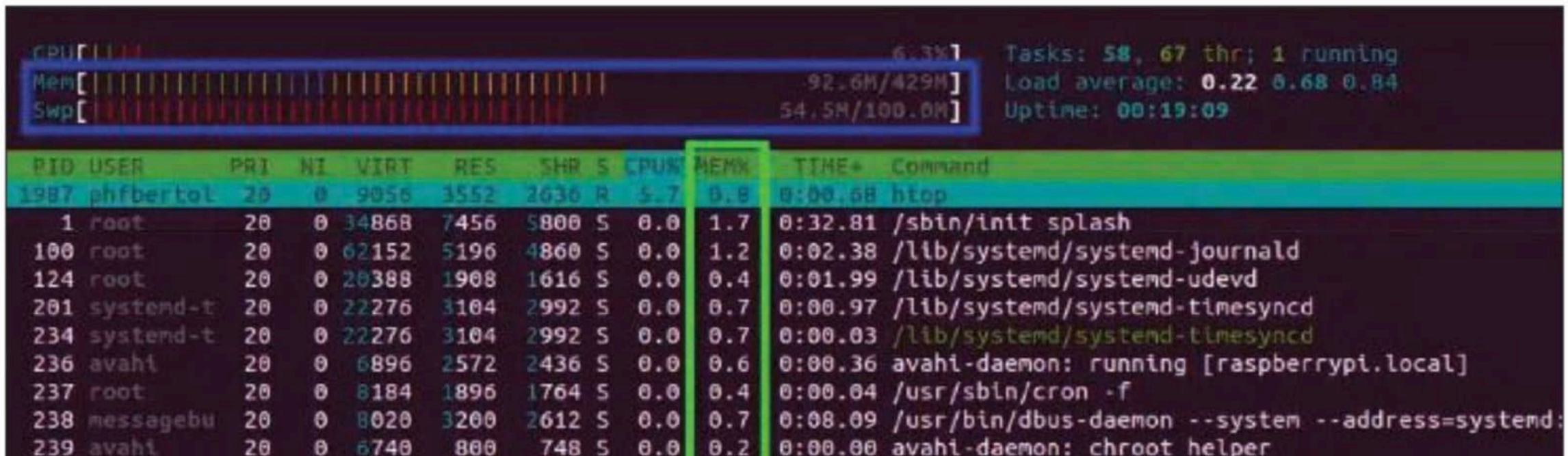

Mem[92.6M/429M]
Swp[54.5M/100.0M]
Tasks: 58, 67 thr; 1 running
Load average: 0.22 0.68 0.84
Uptime: 00:19:09
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
1987 phfbertol 20 0 9056 3552 2636 R 5.7 0.8 0:00.68 htop
1 root 20 0 34868 7456 5800 S 0.0 1.7 0:32.81 /sbin/init splash
100 root 20 0 62152 5196 4860 S 0.0 1.2 0:02.38 /lib/systemd/systemd-journald
124 root 20 0 20388 1908 1616 S 0.0 0.4 0:01.99 /lib/systemd/systemd-udevd
201 systemd-t 20 0 22276 3104 2992 S 0.0 0.7 0:00.97 /lib/systemd/systemd-timesyncd
234 systemd-t 20 0 22276 3104 2992 S 0.0 0.7 0:00.03 /lib/systemd/systemd-timesyncd
236 avahi 20 0 6896 2572 2436 S 0.0 0.6 0:00.36 avahi-daemon: running [raspberrypi.local]
237 root 20 0 8184 1896 1764 S 0.0 0.4 0:00.04 /usr/sbin/cron -f
238 messagebu 20 0 8020 3200 2612 S 0.0 0.7 0:08.09 /usr/bin/dbus-daemon --system --address=systemd:
239 avahi 20 0 6740 800 748 S 0.0 0.2 0:00.00 avahi-daemon: chroot helper
251 root 20 0 40116 4096 3420 S 0.0 0.9 0:00.83 /usr/libexec/polkitd --no-debug
281 root 20 0 2788 1472 1340 S 0.0 0.3 0:00.05 /usr/sbin/dhcpcd -b -q
291 root 20 0 40116 4096 3420 S 0.0 0.9 0:00.00 /usr/libexec/polkitd --no-debug
295 root 20 0 25768 2632 2392 S 0.0 0.6 0:00.35 /usr/sbin/rsyslogd -n -iNONE
305 root 20 0 40116 4096 3420 S 0.0 0.9 0:00.22 /usr/libexec/polkitd --no-debug
308 root 20 0 13248 4640 4236 S 0.0 1.1 0:02.55 /lib/systemd/systemd-logind
313 nobody 20 0 5296 1884 1868 S 0.0 0.4 0:00.05 /usr/sbin/thd --triggers /etc/triggerhappy/trigg
318 root 20 0 25768 2632 2392 S 0.0 0.6 0:00.09 /usr/sbin/rsyslogd -n -iNONE
319 root 20 0 25768 2632 2392 S 0.0 0.6 0:00.02 /usr/sbin/rsyslogd -n -iNONE
320 root 20 0 64872 4884 4068 S 0.0 1.1 0:01.23 /usr/libexec/udisks2/udisksd
321 root 20 0 25768 2632 2392 S 0.0 0.6 0:00.16 /usr/sbin/rsyslogd -n -iNONE
324 root 20 0 11776 2204 2152 S 0.0 0.5 0:00.11 /sbin/wpa_supplicant -u -s -O /run/wpa_supplicar
355 root 20 0 64872 4884 4068 S 0.0 1.1 0:00.00 /usr/libexec/udisks2/udisksd
358 root 20 0 64872 4884 4068 S 0.0 1.1 0:00.08 /usr/libexec/udisks2/udisksd
371 root 20 0 64872 4884 4068 S 0.0 1.1 0:00.00 /usr/libexec/udisks2/udisksd
372 root 20 0 55732 3572 3228 S 0.0 0.8 0:00.85 /usr/sbin/ModemManager
373 root 20 0 29516 4336 4188 S 0.0 1.0 0:00.29 /usr/sbin/cupsd -l
392 root 20 0 37328 3452 3096 S 0.0 0.8 0:00.22 /usr/sbin/lightdm
395 root 20 0 55732 3572 3228 S 0.0 0.8 0:00.00 /usr/sbin/ModemManager
397 root 20 0 37328 3452 3096 S 0.0 0.8 0:00.00 /usr/sbin/lightdm
405 root 20 0 55732 3572 3228 S 0.0 0.8 0:00.09 /usr/sbin/ModemManager
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10

FIGURE 4
RAM and swap memory usage in the htop tool.

SWAP MEMORY USAGE ANALYSIS

First, let's define what memory swap is. Briefly, memory swap is an operation performed by the operating system (including embedded Linux) to move less frequently used data from RAM to a mass storage (such as SSD, micro-SD card, HDD, and so on). This frees up space in RAM for general operating system use, and allows data stored on mass storage to be reloaded into RAM if one or more processes need them. Thus, memory swap helps maintain a healthy amount of available RAM, resulting in faster operation of frequently used processes.

That said, memory swap is a must-have operation for any operating system, especially those that run in SoC or SiP with RAM limitations. This is common in the case of embedded Linux systems, since adding more RAM increases hardware costs, and large amounts of RAM should not be expected in embedded systems that need more and more competitive costs.

Memory swap comes at a cost, however—it causes general sluggishness. The speed of reading, writing, and communicating with RAM is much faster than with mass storage, especially HDDs. Each swap operation is limited by the mass storage's overall speed (read, write, and communication), making it extremely slow from the OS's perspective. Then, it doesn't matter how fast a CPU can be; if the data needed for processes is stored in a slow-access memory, everything will slow down too. **Figure 5** shows why memory swap to a mass storage memory is slow.

To check the amount of swap memory used on embedded Linux, the quickest methods are to use tools like htop or free, as discussed previously in this article for CPU and RAM usage analysis. Manually or automatically monitoring swap usage over time can give a big picture on how aggressively RAM is being consumed on a specific embedded Linux OS-based solution.

Because memory swap is a slow operation, avoiding memory swap is recommended to achieve better overall performance on embedded Linux OS-based solutions. To minimize memory swap occurrences in embedded Linux OS, it's important to understand its main causes. The three most common causes of frequent memory swap in embedded Linux OS are low amount of available RAM, unoptimized memory swap configuration, and high RAM usage by specific processes.

Low RAM Amount Available: When the amount of available RAM is below the solution RAM memory requirements, memory swap tends to occur frequently, significantly slowing down the operating system as a whole.

Unoptimized Memory Swap Configuration: Memory swap is triggered once free RAM reaches a certain threshold. In embedded Linux OS, this is controlled by the swappiness parameter / configuration, which typically can be found in the /proc/sys/vm/swappiness file. This file contains a number ranging from 0 to 100, indicating the percentage of free RAM at which memory swap should activate. (Setting it to zero disables swap and setting it to 100 allows frequent memory swap operations.) Thus, higher swappiness values (closer to 100) mean memory swap will begin even with a considerable amount of available free RAM. This can be unnecessary in many real-world scenarios, causing a general slowdown in embedded Linux OS. For instance, if swappiness is set to 80, swap begins when RAM is at 20% free. Moderate swappiness values (around 60) lead to good balance in swap usage, without causing significant slowdowns to the operating system.

High RAM Usage by Specific Processes: Occasionally, due to suboptimal development choices, specific processes may allocate more RAM than expected, consuming more of the system's RAM and increasing memory swap

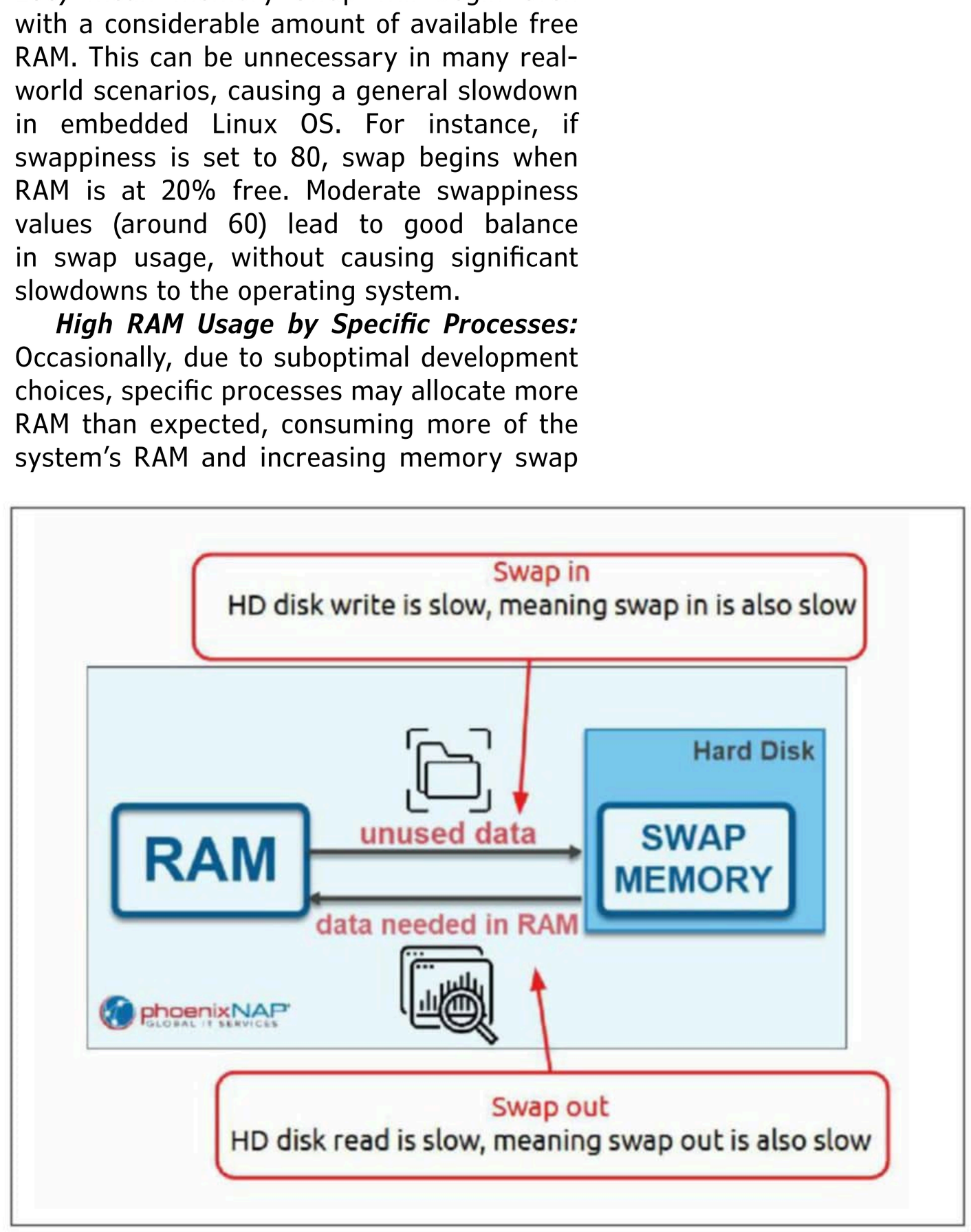


FIGURE 5
Illustration of why memory swap is slow. (Image source: phoenixNAP Global IT Services [2])

ABOUT THE AUTHOR

Pedro Bertoleti is a Brazilian embedded systems engineer and publisher of a developer website called Embarcados – www.embarcados. com.br. He has strong experience in development with PIC, ESP32, ARM Cortex M0 and M3 and Rabbit RCM22xx microcontrollers. Bertoleti also has strong knowledge and experience in IoT projects and Embedded Linux Embedded Systems development.

usage. This situation often arises when a solution developed for a personal computer or server environment (likely with more RAM available) is ported to embedded Linux OS, without taking in consideration the amount of RAM available in target hardware.

MEMORY THRASHING ANALYSIS

Cutting to the chase, memory thrashing is a direct consequence of using too much memory swap. As previously discussed, the bottleneck with memory swap is the slower read, write, and communication speeds associated with mass storage memory used for storing memory swap data.

When embedded Linux OS reaches a state where free RAM is so limited that constant memory swapping is required, frequent interaction with mass storage ensues. Since this interaction is slow, the system may reach a point at which it spends more time performing memory swaps than using the data read and written in memory swap operations, leading to an extreme slowdown in embedded Linux OS. This situation is commonly known as "memory thrashing."

To put it in perspective, imagine digging a deep hole using only a spoon. The spoon (representing RAM) can carry only a small amount of material such as data, buffers, so the person digging (representing the operating system) must make frequent trips to a storage container (mass storage) to empty the spoon (memory swap) and then resume digging. Given the spoon's limited capacity, this emptying process takes significant time relative to the main task of digging, leading to excessive time spent swapping or memory thrashing.

To diagnose memory thrashing, certain RAM usage metrics in embedded Linux should be monitored closely. These metrics are available through the same tools covered so far in this article—the htop, sar, and free tools.

Some indicators of memory thrashing in embedded Linux OS are noted below:

- High swap memory usage.
- Swap memory usage is increasing substantially over time.
- Rising I/O wait time (known as IOWait in embedded Linux). The IOWait reflects the percentage of time one or more CPUs spend waiting for responses from an I/O device, such as mass storage memory. Therefore, the more memory swap occurs, the greater the IOWait value will be.

Since memory thrashing is highly undesirable in embedded Linux OS, it's essential to minimize memory swap. This can be done by monitoring and mitigating the causes of memory swap discussed above.

MEMORY LEAK ANALYSIS

In terms of ensuring good performance, a memory leak is one of the worst enemies that embedded Linux can have. A "memory leak" is a serious flaw in software, where memory allocated dynamically by the software is not released after use, rendering that portion of RAM unavailable for any other program. If this issue occurs in a frequently-called routine, it will quickly exhaust the system's free RAM, leading to severe performance issues—forcing memory swap, causing memory thrashing, and so on—and potential system failures. In extreme cases, it may even crash the entire system, requiring a restart.

A classic case of a memory leak is when a C program allocates memory dynamically using the `malloc()` function but fails to release it afterward with the `free()` function. When this is done one hundred times a second, here comes one of the most common embedded software developer's nightmares.

To detect memory leaks in embedded Linux, it's recommended to monitor RAM usage periodically (system-wide or by individual processes), either manually or automatically. This can be done using tools discussed previously: the htop, ps, and free tools. If RAM usage of a specific process steadily increases over time, it suggests a memory leak.

CONCLUSION

This article concerns techniques and tools for analyzing the performance of embedded Linux, from the perspective of CPU and RAM use.

It is possible to perform an analysis under these aspects to determine whether processes are using more computing resources than expected. In this way, the developer will be able to understand the possible reasons why the performance of embedded Linux in a given solution is lower than what is expected, and also determine which processes are responsible.

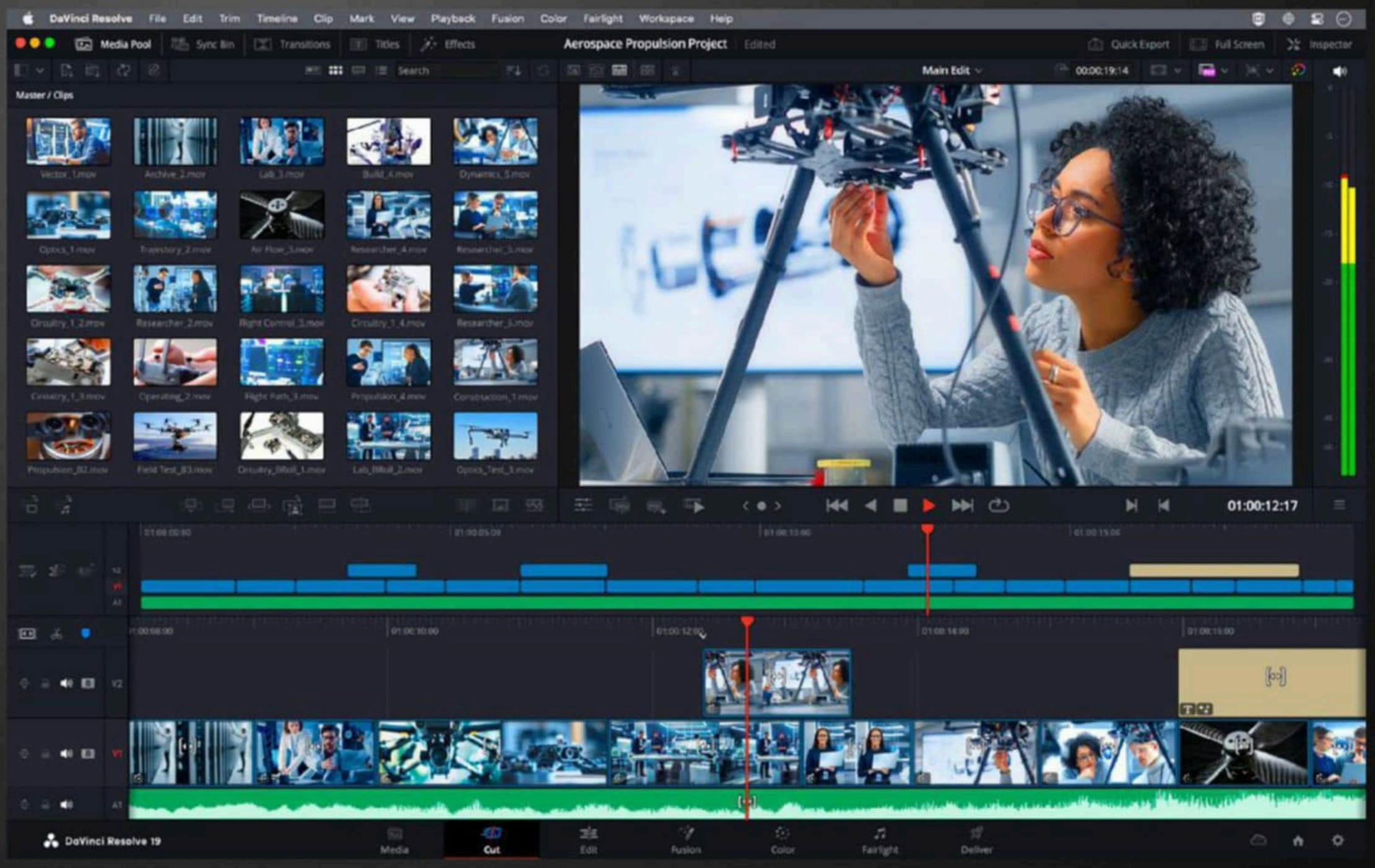
Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] to [2] as marked in the article can be found there.

RESOURCES

Linux | www.linux.com

FREE Download Now!
Mac, Windows and Linux

Blackmagicdesign



Edit and color correct using the same software used by Hollywood, for free!

DaVinci Resolve is Hollywood's most popular software! Now it's easy to create feature film quality videos by using professional color correction, editing, audio and visual effects. Because DaVinci Resolve is free, you're not locked into a cloud license so you won't lose your work if you stop paying a monthly fee. There's no monthly fee, no embedded ads and no user tracking.

Editing, Color, Audio and Effects!

DaVinci Resolve is the world's only solution that combines editing, color correction, visual effects, motion graphics and audio post production all in one software tool! You can work faster because you don't have to learn multiple apps or switch software for different tasks. For example, just click the color page for color, or the edit page for editing! It's so incredibly fast!

Professional Editing

DaVinci Resolve is perfect for editing sales or training videos! The familiar track layout makes it easy to learn, while being powerful enough for professional editors. You also get a library full of hundreds of titles, transitions and effects that you can add and animate! Plus, DaVinci Resolve is used on high end work, so you are learning advanced skills used in TV and film.

Creative Color Correction

DaVinci Resolve's color page is Hollywood's most advanced color corrector and has been used on more feature films and television shows than any other system! It has exciting new features to make it easier to get amazing results, even while learning the more advanced color correction tools. There's PowerWindows™, qualifiers, tracking, advanced HDR grading tools and more!

Designed to Grow With You

DaVinci Resolve is designed for collaboration so as you work on larger jobs you can add users and all work on the same projects, at the same time. You can also expand DaVinci Resolve by adding a range of color control panels that let you create unique looks that are impossible with a mouse and keyboard. There's also edit keyboards and Fairlight audio consoles for sound studios!

DaVinci Resolve 19 ..... **Free**
DaVinci Resolve Micro Color Panel ..... **Only $509**

Learn the basics for free then get more creative control with our accessories!

www.blackmagicdesign.com

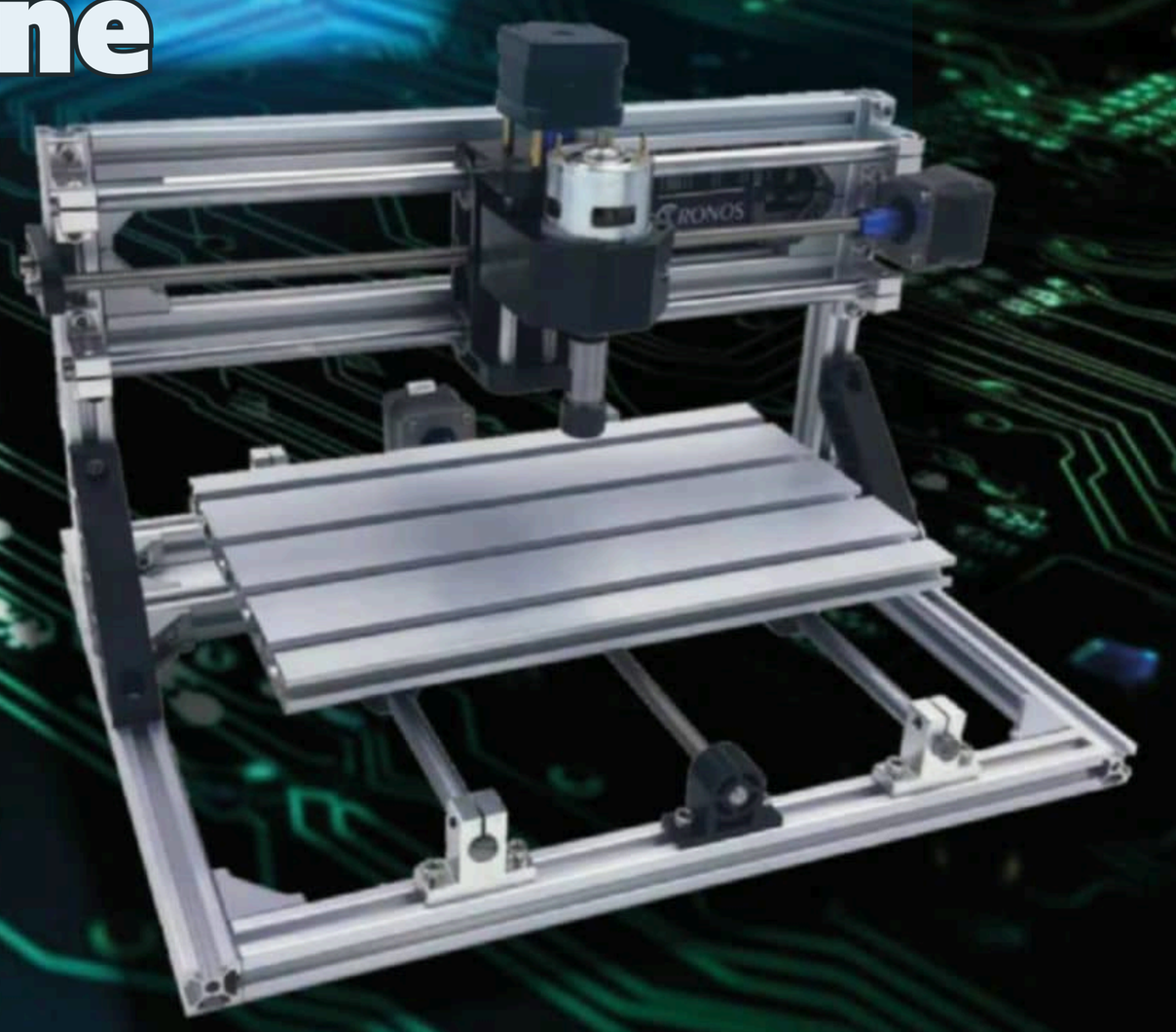
Learn More!

NO SUBSCRIPTIONS • NO ADS • NO USER TRACKING • NO AI TRAINING

FEATURES

Create Your Own PCBs with a CNC Milling Machine

Using KiCad, CopperCAM, and Candle Software



By
Matthew Oppenheim

A Computer Numerical Control or CNC machine is a computer-controlled device that uses pre-programmed software to cut, shape, and create parts and prototypes. In this article, Matthew demonstrates how to use a small CNC milling machine at home to make a simple two-sided PCB, using KiCad, CopperCAM, and Candle software.

A Computer Numerical Control (CNC) machine is controlled by a computer and uses pre-programmed software to cut and shape parts and prototypes. They are used with Computer Aided Design (CAD) and Computer Aided Manufacture (CAM) software. In this article I describe my workflow for manufacturing a simple, two-sided printed circuit board from a blank piece of copper-clad FR4, using a Genmitsu 3018 PROVer V2 CNC milling machine [1].

Why, in the age of cheap PCB production, would anyone take the time to mill out their own PCB at home? I asked myself this question a few times during this project.

I don't suggest that it is practical to mill a fully functional PCB that has a complex design at home, one with many vias, or component footprints with finely spaced surface mount pads. Some artisans do manage to do this and I credit them for it. However, modifying a complex PCB design so that it retains the board outline, mounting holes, and footprints required for the board connectors does not take long, and allows us to manufacture a "looks like" prototype that enables us to check two things:

- That the PCB fits inside the intended case correctly.
- That any connectors on the board align correctly with the openings in the case.

For this article, I milled a complete, two-sided PCB that connects to a LILYGO T-Embed module [2] using 0.1" spaced header pins. T-Embed is an IoT embedded panel designed for programmable development. LILYGO provides 3D and reference drawings for their products. The completed and populated board is shown in **Figure 1**. I used the completed assembly as a part of an assistive technology project called "handshake" [3]. The project is designed to allow people with limited dexterity to interact with assistive technology devices and software.

The board enables the T-Embed to act as an access switch to operate assistive technology. The standard interface for these assistive technology devices is a 3.5mm plug and socket. When the tip and sleeve of the

plug are shorted, a switch closure is detected by the hardware that is being controlled.

The PCB I built has a 3.5mm socket that can short the tip and sleeve of an attached 3.5mm plug, using an opto-isolated relay controlled by the T-Embed. I hope to present a more detailed article on this project in the future. See **Figure 2** for the schematic of the PCB.

The workflow for creating the PCB has three steps, which are discussed in detail in the next sections. They are:

1) Creating Gerber and drill files from my PCB design using KiCad [4] software. A Gerber file is a vector image file containing information about a PCB layer.
2) Creating G-code files using CopperCAM [5] software. G code is the programming language most commonly used for CNC and 3D printing. CopperCAM manages the drilling, cutting, and engraving of PCB prototypes.)
3) Sending the G-code to the CNC using Candle [6] software.

CREATING GERBER AND DRILL FILES USING KICAD

I use KiCad for schematic capture and PCB layout. To prepare the PCB for use with CopperCAM, we need to move or copy the board edge line from the Edge.Cuts layer to the front copper F.Cu layer. Why? It is easier to tell CopperCAM that a trace on the top copper layer is the edge cut than to try and align a separate edge-cut layer with the component layers.

The next step needed is to create Gerber and drill files to import into CopperCAM. To create the Gerber files:

1) Go to the menu option: File, Fabrication Outputs. Plot F.Cu, B.Cu.
2) Unselect "Plot footprint text" (see **Figure 3**).
3) Plot

To create the drill file:

1) Go to the menu option: File, Fabrication Outputs, Drill Files.
2) Tick the PTH and NPTH in a single file option; this option defaults to not being selected. All other options are default (see **Figure 4**).
3) Generate Drill File

USING COPPERCAM SOFTWARE

The CNC's spindle is told how fast to spin and how to move, using a language called G-code. CopperCAM software uses the Gerber and drill files that we created with KiCad to calculate the G-code needed to mill out the traces and to drill the holes on the PCB. CopperCAM software is designed for Windows, but will run smoothly on Linux

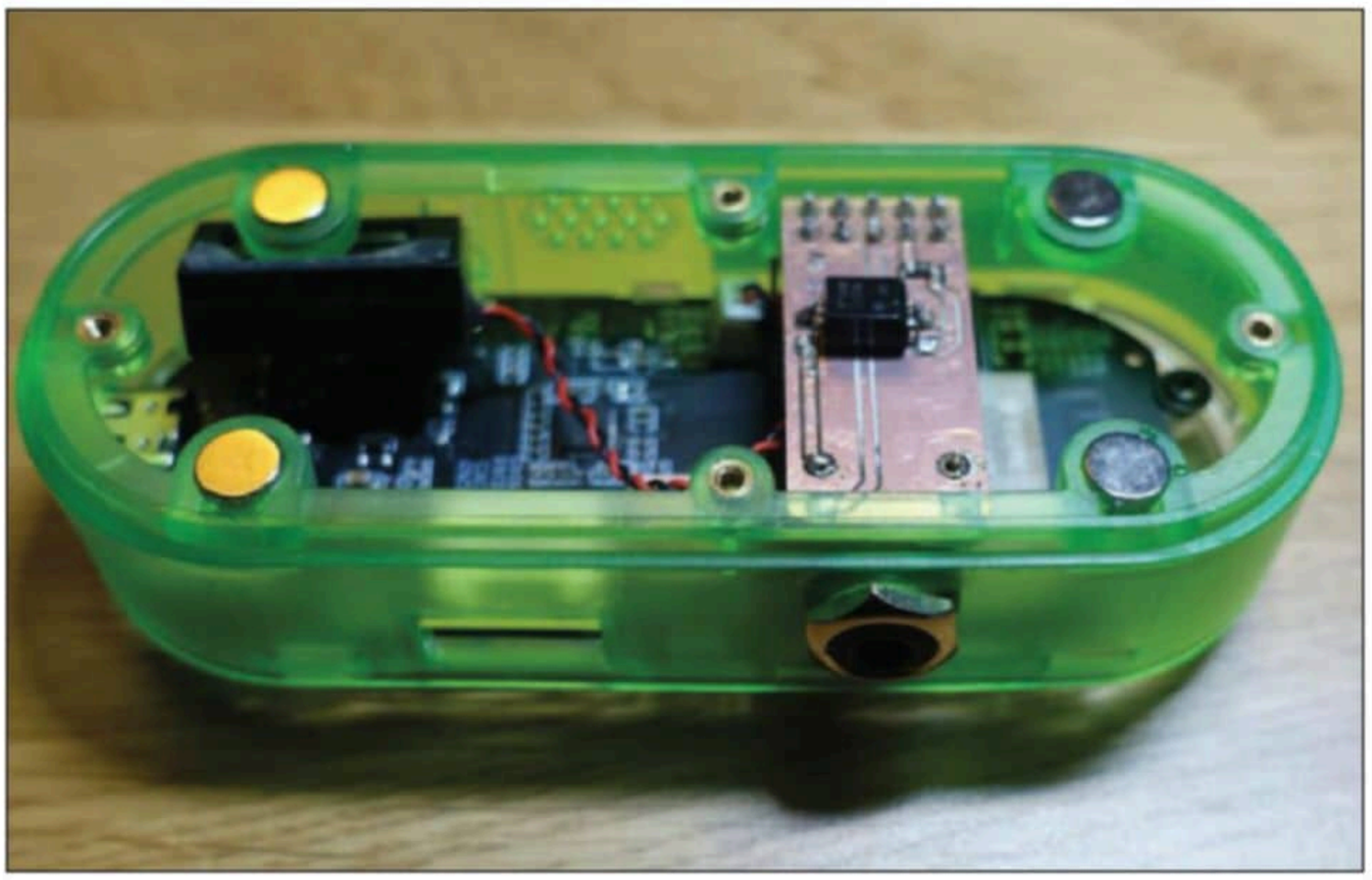


FIGURE 1
Completed PCB connected to a LILYGO T-Embed.

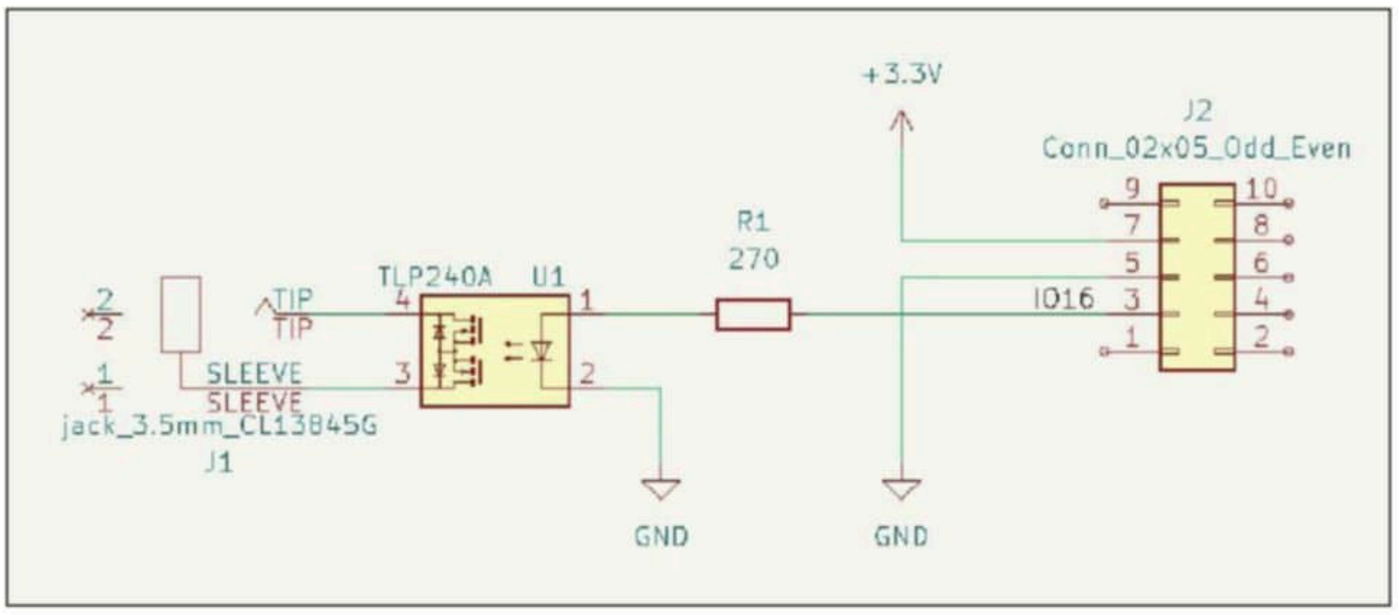


FIGURE 2
Schematic for the access switch adapter board that connects with a LILYGO T-Embed.

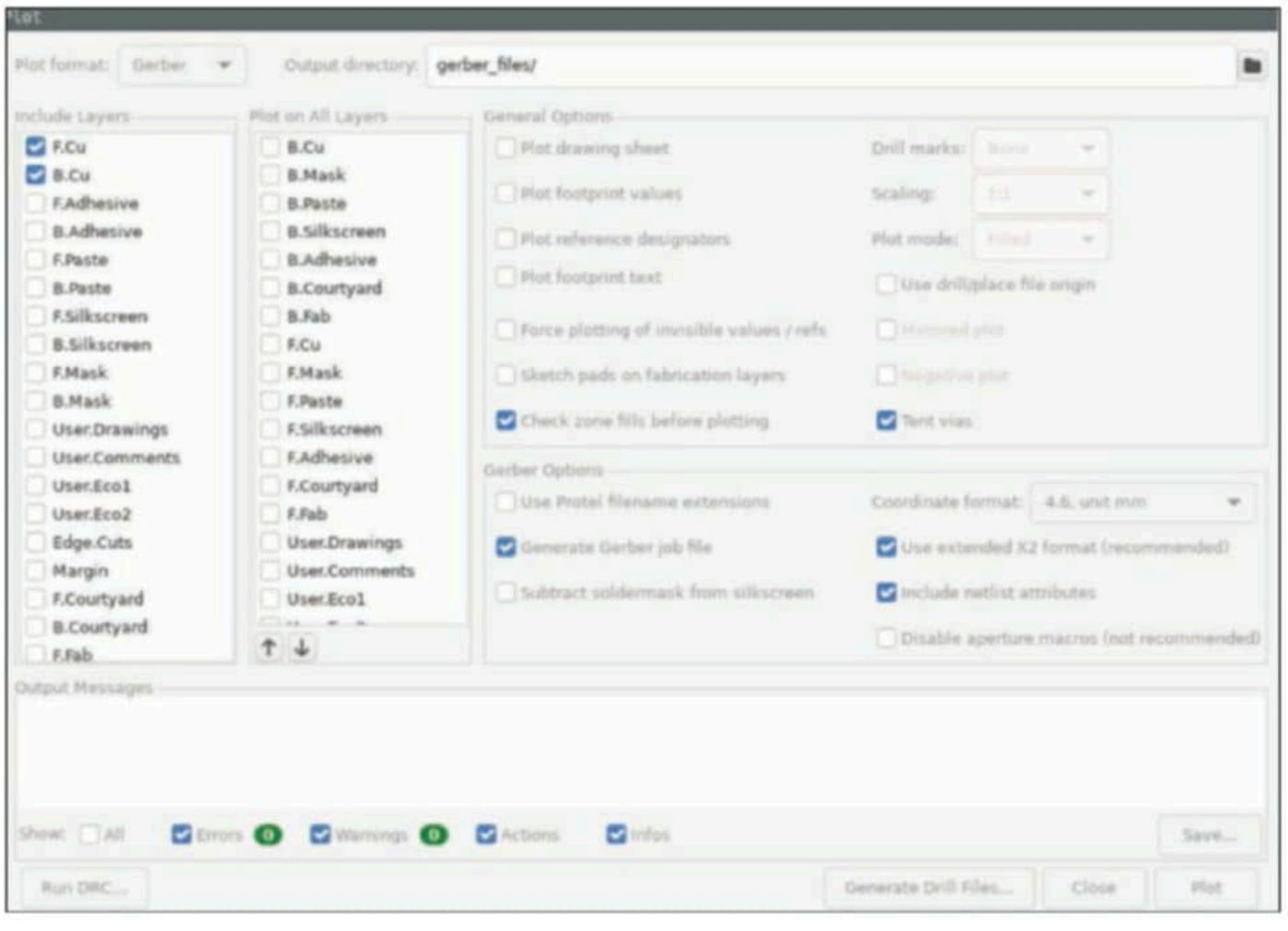


FIGURE 3
KiCad Gerber export options.

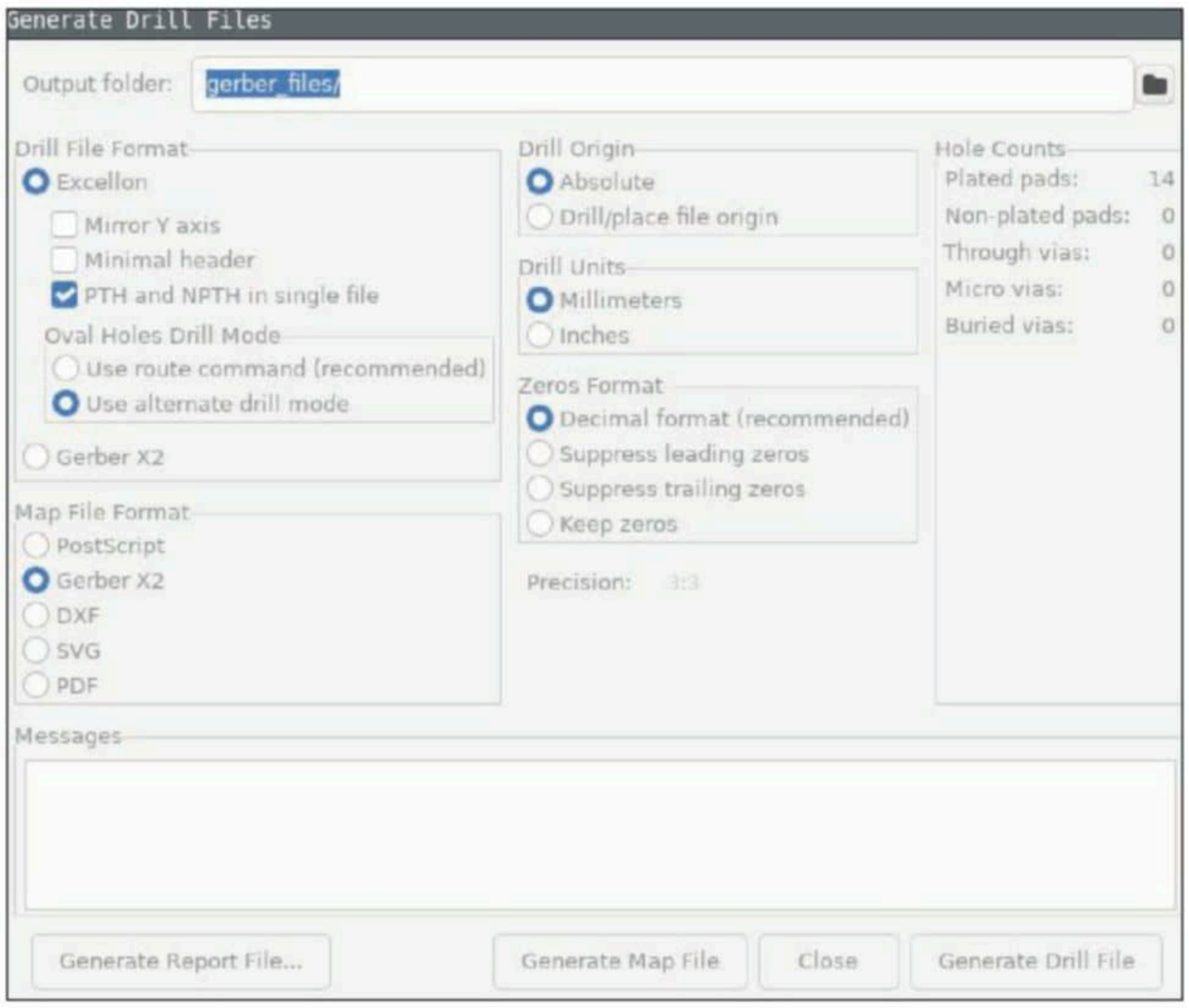


FIGURE 4
Creating drill files using KiCad.

using the Windows emulation software called Wine [7].

A demo version of CopperCAM can be downloaded from galaad.net [8], but is limited in the number of pads for which the software will produce G-code. Unlimited capability, but time-limited licenses, also are available, so that you can fully test the software. I paid for a license, which was €80 (about $84) [9]. The license is "lifetime use anywhere." The other software used in this article is free and open-source.

I don't begrudge paying for software that works well and allows me to own it, but I'm not a fan of having to pay a subscription for software. CopperCAM is not sponsoring me for this article. A colleague mills out his PCBs using the free bCNC [10] software. I had a look at bCNC, but found it less user friendly than CopperCAM.

INSTALL COPPERCAM USING WINE

I'm including details on how to install and use CopperCAM using Linux, because this is how I did it, and I anticipate that a lot of the readers of this esteemed publication also use Linux.

These are the commands necessary to install Wine [7] on Debian 12, mostly copied from a helpful website [11].

First, we need to install the 32-bit version of Wine. Avoid the Flatpak installation, I couldn't get that to work. These are the commands I used to install Wine:

```
sudo mkdir -pm755 /etc/apt/keyrings

sudo wget -O /etc/apt/keyrings/
winehq-archive.key https://dl.winehq.
org/wine-builds/winehq.key

sudo wget -NP /etc/apt/sources.
list.d/ https://dl.winehq.org/wine-
builds/debian/dists/bookworm/winehq-
bookworm.sources

sudo apt update

sudo apt install --install-recommends
winehq-stable

export WINEARCH=win32
```

For whatever reason, to get the 32-bit version of wine to install, I had to remove the existing ".wine" directory in my home directory. Rename the directory for safety instead of removing it,

```
mv ~/.wine ~/wine_backup
```

Now install CopperCAM from whatever directory to which you downloaded the .exe installation:

```
wine coppercam.exe
```

The installation process will ask to install .net. Ok this.

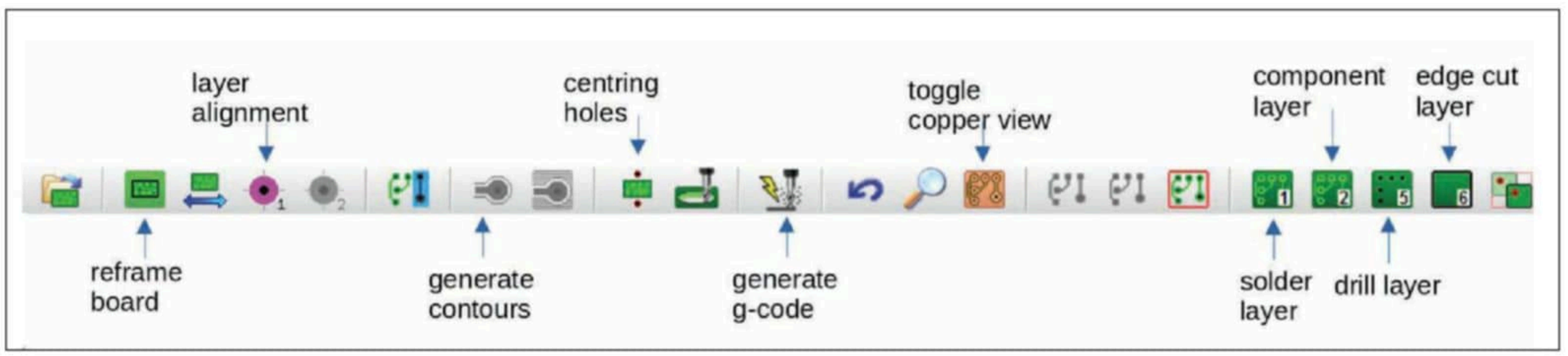


FIGURE 5
Annotated CopperCAM menu bar.

To run the CopperCAM software use this command:

```
wine ~/.wine/drive_c/CopperCAM/
CopperCAM.exe
```

CREATING G-CODE

The menu icons along the top bar of CopperCAM don't have any pop-up tool-tips to clarify what each one does, at least not when running it under Wine on Linux. **Figure 5** is an annotated screenshot of the menu icons, with the tools I used in this article labeled. This is an illustration I wish I'd had when I started!

The best online resource that I found for using CopperCAM to prepare a PCB is a YouTube video, "CopperCAM Tutorial," by Jamesong10 [12]. It is worth watching this video before reading further. The steps to prepare your PCB are given below:

Import the Gerber and Drill Files: First, we need to import the Gerber and drill files that we created in KiCad.

1) Go to the menu option File, Open, new circuit.
2) In the pop-up menu, we need to define which of our Gerber files corresponds with which construction stage of the PCB. Select these options:

- Solder side, select the B.Cu Gerber file. This defines what to mill out on the solder side of the PCB, which is probably the bottom side of your board.
- Component side, select the F.Cu Gerber file. This defines what to mill out on the component side of the PCB, which is probably your top side.

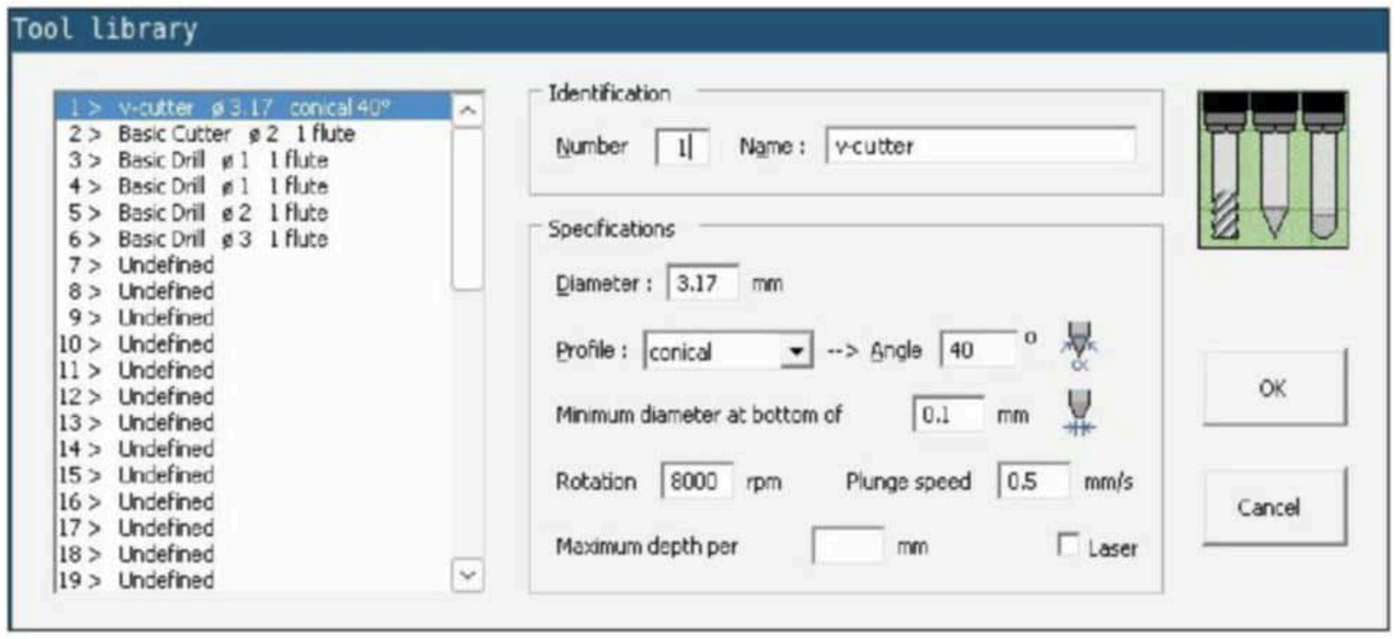
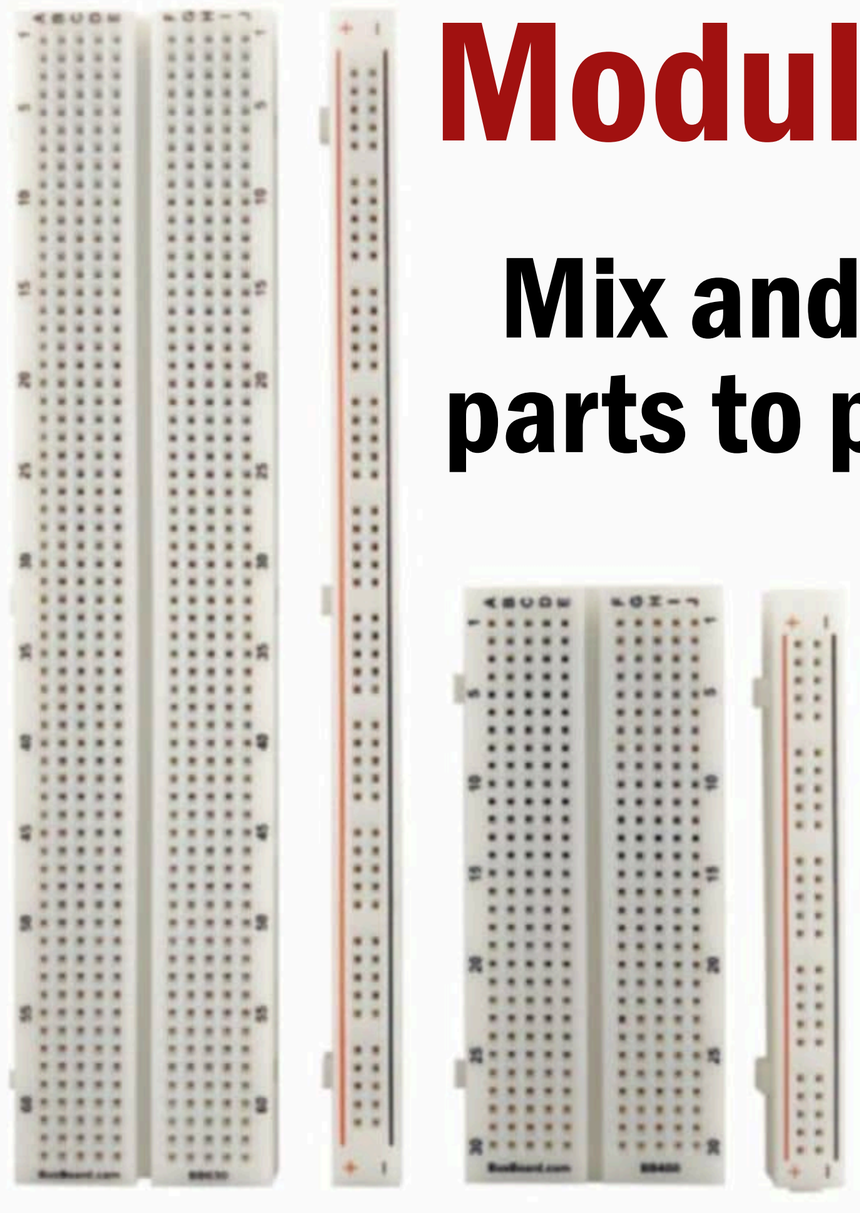


FIGURE 6
Example of CopperCAM tool library setup.



Modular BreadBoards

Mix and match our BreadBoard parts to perfectly fit your project

Start your prototyping with one of our many full-height (*BB830M, BB1460, BB1560M*) or half-height (*BB400M*) breadboards and easily detach and rearrange the power rails and circuit areas to build the perfect layout.

Easily change your layout and add additional circuit area (*BB630, BB300*) and power rail (*BB100R, BB50R*) components as your project grows.

Purchase at your trusted retailer

MOUSER ELECTRONICS DigiKey amazon.com

BPS™ BusBoard Prototype Systems

www.BusBoard.com/BB

- Drills, select the drill file. I use a single drill bit size and drill out any larger holes using a drill press.
- Cut-out, leave blank. This is the outline of the board. Your cut out should be in the F.Cu layer as detailed earlier in this article. We will select the trace to use as the edge cut presently.

3) If you are using Linux, continuously navigating from your Wine directory to the Gerber files directory can become tedious. My tip is to make some soft links in the CopperCAM .wine directory to three locations: Gerber and drill files directory; CopperCAM project file directory; and G-code directory.
4) Once you've imported your Gerber and drill files, you will see your design in CopperCAM. Have a good look at all the pad shapes because sometimes some of these are changed from what you had in KiCad. Right click on any incorrect pads, and select the correct shape and size. From what I've read, there are different standards for Gerber files; this leads to the occasional confusion on pad shape when importing Gerber files.
5) Right click on the trace that you want to be your edge cut, and select this option as "Define as cutting contour" from the menu that appears. CopperCAM is pretty good at working out what the outline is. Select that the milling tool that cuts out the board will go outside this path, or you'll end up with a smaller board than you anticipated.

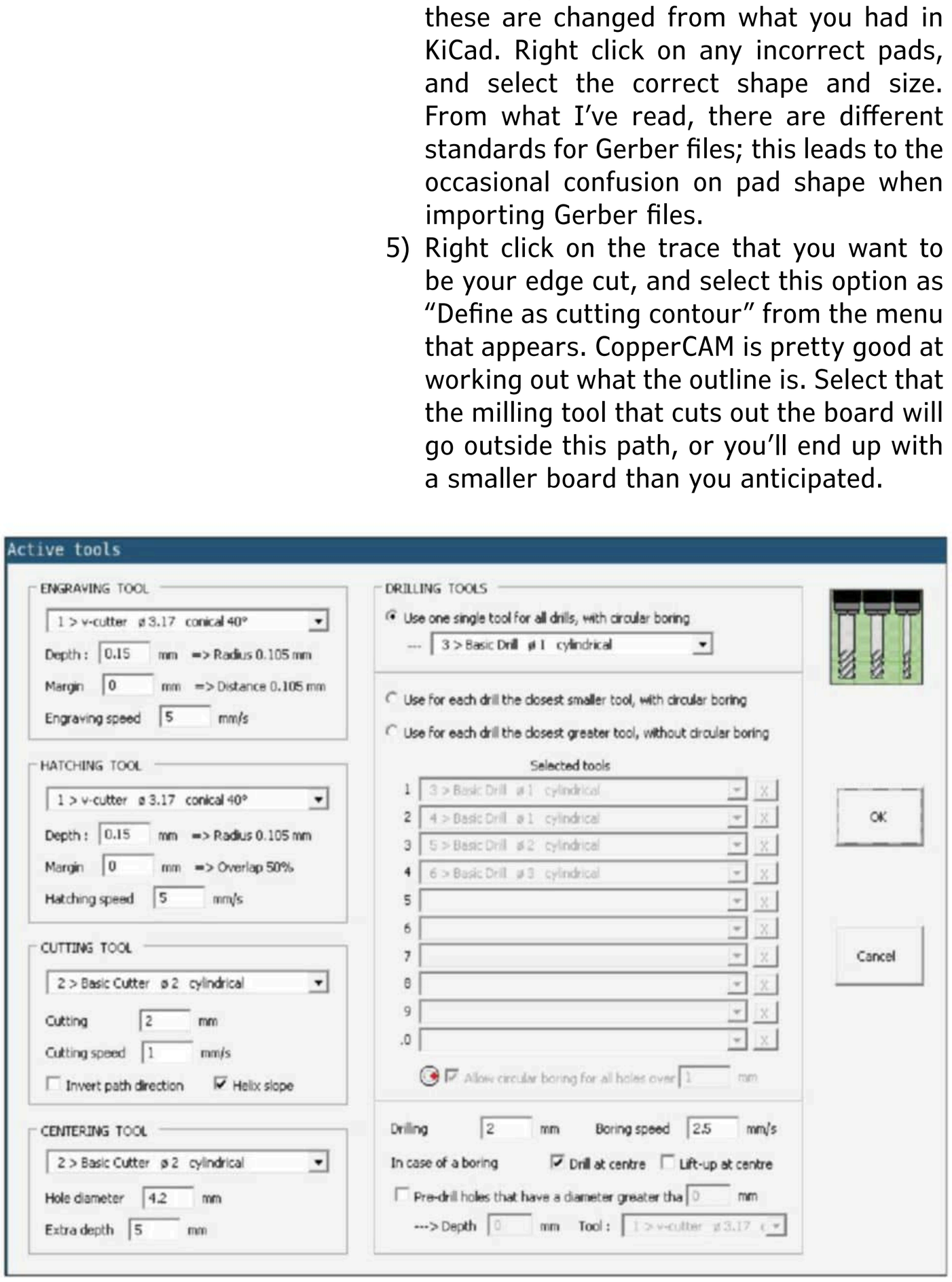

Active tools
ENGRAVING TOOL
1 > v-cutter ø 3.17 conical 40°
Depth : 0.15 mm => Radius 0.105 mm
Margin 0 mm => Distance 0.105 mm
Engraving speed 5 mm/s
HATCHING TOOL
1 > v-cutter ø 3.17 conical 40°
Depth : 0.15 mm => Radius 0.105 mm
Margin 0 mm => Overlap 50%
Hatching speed 5 mm/s
CUTTING TOOL
2 > Basic Cutter ø 2 cylindrical
Cutting 2 mm
Cutting speed 1 mm/s
Invert path direction
Helix slope
CENTERING TOOL
2 > Basic Cutter ø 2 cylindrical
Hole diameter 4.2 mm
Extra depth 5 mm
DRILLING TOOLS
Use one single tool for all drills, with circular boring
3 > Basic Drill ø 1 cylindrical
Use for each drill the closest smaller tool, with circular boring
Use for each drill the closest greater tool, without circular boring
Selected tools
1 3 > Basic Drill ø 1 cylindrical
2 4 > Basic Drill ø 1 cylindrical
3 5 > Basic Drill ø 2 cylindrical
4 6 > Basic Drill ø 3 cylindrical
5
6
7
8
9
.0
Allow circular boring for all holes over 1 mm
Drilling 2 mm Boring speed 2.5 mm/s
In case of a boring Drill at centre Lift-up at centre
Pre-drill holes that have a diameter greater tha 0 mm
---> Depth 0 mm Tool : 1 > v-cutter ø 3.17
OK
Cancel

FIGURE 7
Example of CopperCAM tool settings.

6) If the top and bottom layers become misaligned, the layers can be realigned using the layer alignment tool (Figure 5). Click on a pad on one side of your board, activate the other layer, and click on the corresponding pad on that layer. Watch the YouTube video mentioned earlier [12] for a demonstration of how to do this.

Define Drill Holes for Side 2 of the PCB: The next step is to define where two drill holes go that are used to align the board when you flip it over to mill the second side.

1) I use M4 acrylic bolts. I found a 4.1mm hole to be the right size. I use acrylic bolts for the unlikely scenario where I miscalculate where the outline edge cut goes, and end up shaving the bolt head with the CNC cutting tool when I cut out the finished PCB. Acrylic doesn't mind having a bit of it shaved off, whereas steel does. Guess how I found this out? I found that 4.1mm holes allow for a snug fit between the hole and an M4 acrylic bolt.
2) Offset an M4 bolt hole by at least 1.6mm from the edge of the board to allow for the extra radius of the bolt head. This works if you are using a 2mm diameter cutter to route out the board.
3) Place the two drill holes around about the center of your board. Then accurately recenter the board around these holes by using the menu option "Machining, Add centering holes, Recenter circuit."
4) Now define the bottom hole as your coordinate origin by selecting the Reframe button on the menu.
5) Click on the crosshairs icon, select "Plot manually," then click on the bottom hole. This hole should flash red. In earlier versions of the software, a white cross was plotted over the hole. In the version of CopperCAM that I used for this article (Version: 20/09/2024), the white cross did not appear on the display.

If we set up the two drill holes in this way, when the board is flipped to machine the second side, the board is mirrored around the origin. This means that the origin is in the same place for machining both sides of the board. If we use four holes and define, for example, the bottom left hole as the origin, after flipping the board, the origin will move. Unless you keep track of this and redefine the origin for each side of the board, the traces will be not be milled on top of each other. Guess how I found this out?

Define the Settings for Milling and Drilling: Now we need to define the settings for each of the tools that are used to mill and drill the PCB.

Go to the menu option Parameters, Tool library. To create a PCB we need an engraver for milling out the traces and pads, a cutter for cutting out the board and at least one drill for drilling through holes. I use a 40° conical cutter for milling out the traces and pads, a 2mm cylindrical basic cutter for cutting out the PCB and a 1.0mm drill for all holes. Please see **Figure 6** for a screenshot of my Tool library settings. Obviously, you need to set this up with the parameters of the tools that you use.

Assign Tools to Stages of PCB Manufacture: We now need to assign these tools to each of the stages of PCB manufacture. In the menu, go to Parameters, Active tools. The first time that you use CopperCAM, the options will have default settings. For the little CNC that I own, the default movement speed for each of the tools is too high, leading to poor results. I at least halve the default movement speeds. The Engraving Tool, Depth value is critical. This defines how deep the tool will cut to mill out you pads and traces. Too shallow and there will be a short between your trace and the surrounding copper. Too deep and the pad shapes will be distorted and the traces may be completely milled out. I tested different values and settled at 0.15mm for the v-cutter that I use. A screenshot of the settings that I use is shown in **Figure 7**.

Create G-Code for CNC Spindle: Now, we need to create the G-code that tells the CNC spindle where to move and how fast to spin.

1) Click on the Generate G-code icon (Figure 5). The first time you use this option, a screen comes up with information about the G-code standard that is to be used. Just click on OK.
2) In the next screen, we define and export G-code for each of five construction

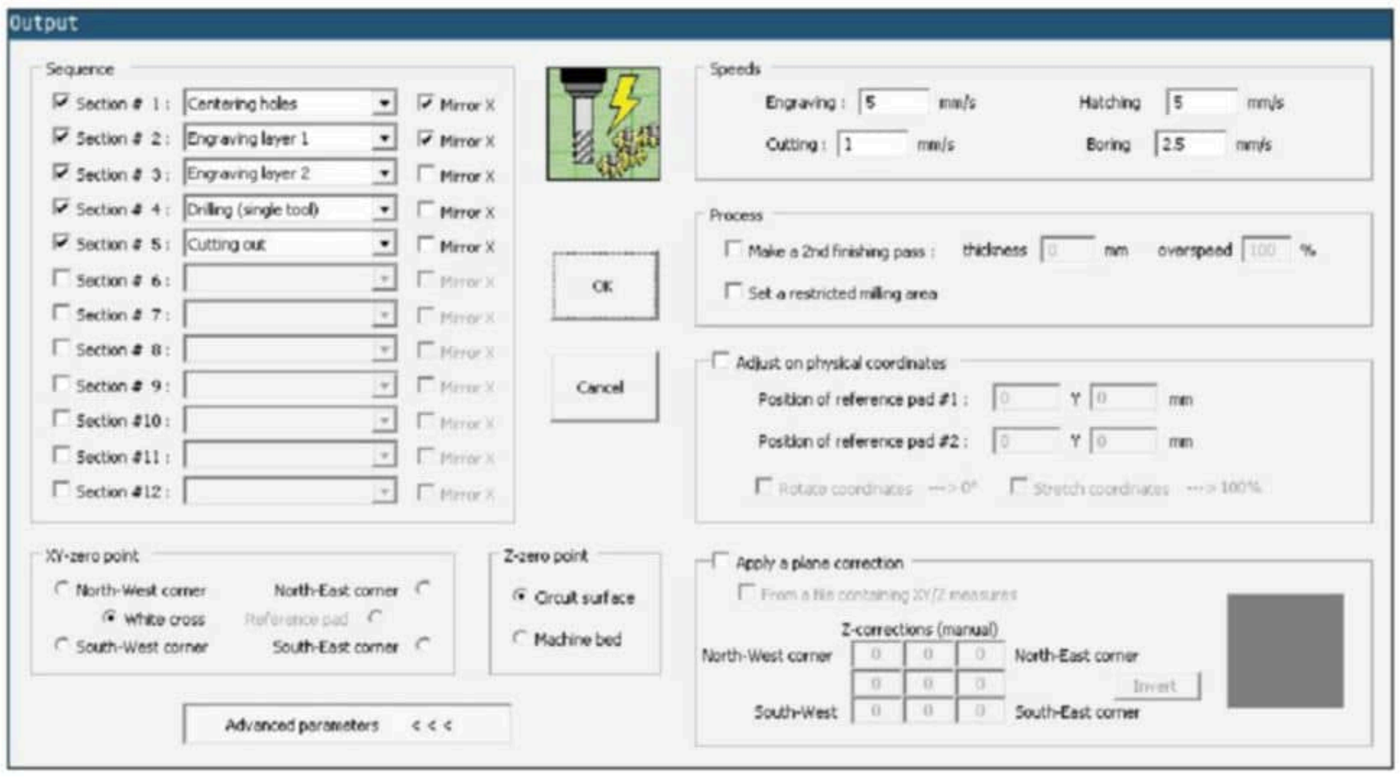


FIGURE 8 CopperCAM screenshot showing G-code setup.

stages. I created a screenshot showing all five stages at once (**Figure 8**). Only select one stage at a time when you create your g-code.

3) Things to check when exporting the G-code:
   - Click the "Mirror x" for the first two stages, which create the centering holes and engrave the bottom layer.
   - Click on "White cross" for the XY-zero point. The white cross is the bottom drill hole that we defined as the coordinate origin earlier. Note that the version of CopperCAM I used for this article did not display this white cross, though an earlier version of the software does.

4) Select each construction stage shown in Figure 8, then click "OK." I use the default naming convention for the G-code files, since this name includes the manufacturing step number and a brief description of what that step is.

Bash Script for Editing G-Code File: Now we have the G-code to run our CNC. Almost. As with Gerber files, there is some flexibility as to what valid G-code is. I had to edit the G-code files that CopperCAM outputs using a small bash script to make it run on Candle software (**Listing 1**). This script uses the `sed` command to edit the G-code file.

Run the script with the G-code file as an argument. To run this script on all G-code files in the directory containing the script, use the bash command:

```
find . -name '*.iso' -exec ./clean_
gcode.sh {} \;
```

This will also run the script in any sub-directories. To limit the script to the current directory use this bash command:

```
find . -name '*.iso' -maxdepth 1
-exec ./clean_gcode.sh {} \;
```

We now have the G-code to send to our CNC to mill and drill our PCB. Before milling and drilling, however, we need to set up the CNC milling machine.

CNC MACHINE SETUP

My father trained as a machinist, and kept a traditional lathe and milling machine in his basement. I spent some hours learning the basics of machining using them. Setting up and clamping the material in the milling machine can be one of the trickiest steps. An old friend of mine, who is now a physics professor, has a much larger and more powerful CNC than I do. I spent one happy morning helping him build a laser on his kitchen table, by attaching various components to an aluminium base plate he machined in his garage with his CNC.

I now follow his advice of sticking the blank PCB to a flat piece of 5mm acrylic with double-sided tape. This keeps the PCB flat, so the milling tool stays at the same cutting depth when the traces are cut out. I hold the acrylic sheet in a low-profile vice [13]. It is possible to secure the acrylic sheet to the base plate of the CNC, but it is a bit more fiddly than using a vice, and there is a risk of running the cutting tool into clamps that protrude above the surface of the PCB.

Make sure to put some double-sided tape under the milling area of your PCB before you machine the second side, because it is the only was you can secure the board when you do the final edge cut process. **Figure 9** shows a completed PCB ready to remove from my Genmitsu CNC. The board is attached with double-sided tape, as described above.

Take a minute to think through the necessary safety precautions when running a CNC. Always wear safety glasses. If your tool path is incorrect and the tool collides with a clamp or the base plate, it could shatter. The dust kicked up from carving the FR4 in a PCB is basically powdered glass, which you don't want in your lungs, so wear a filter mask. Use a vacuum cleaner to clean up the dust. Keep an eye on the machine at all times when it is operating, and be ready to hit the emergency stop.

LISTING 1
Bash script to edit G-code produced by CopperCAM, so that it runs on Candle software.

```
# Clean g-code so that it is accepted by Candle software
# run ./clean_code.sh <filename>
# the input is edited in place
#!/bin/bash
# Use the sed command to edit the file
sed -i '
# Remove lines starting with the % symbol
/^%/d
# Remove lines starting with the T symbol - these are tool definitions
/^T/d
# Remove lines starting with the ( symbol - these are comments
/^(/d
# operate on the file name supplied at run-time
' "$@"
```

USING CANDLE TO CNC MACHINE

Candle software sends the G-code to the CNC through a USB cable. The software shows the path that the tool tip will take for the G-code file. A panel on the right side of the screen allows you to control the CNC spindle position and to set the origin for either the x, y coordinates or the z position of the tool-tip.

The x, y origin and the z-axis origin controls tell Candle where the origin point of the G-code file will physically be on the PCB. **Figure 10** shows these controls with labels. It is important to get the origin point right, or you'll end up milling somewhere you didn't want to, potentially driving your tool into the vice or CNC bed. Guess how I found this out?

To set the x, y axis origin position, manually move the tool tip to where you want the bottom hole drilled for your alignment holes. This is the position we defined as the coordinate origin in CopperCAM. Click on the x, y origin button on Candle. You will see the x and y coordinates in the top row on the right hand panel go to zero.

The origin position for the z-axis is where the tip of the tool touches your PCB. Obviously, you need to change the tool in between some of the milling stages, so you will have to reset the z-axis origin after each tool change. Use the z-axis knob on your CNC to lower the tool until the tip just touches your PCB. How do you know when this is? Use a multimeter with two bulldog clips, one on the tool and one clipped onto your PCB. The instant you get continuity

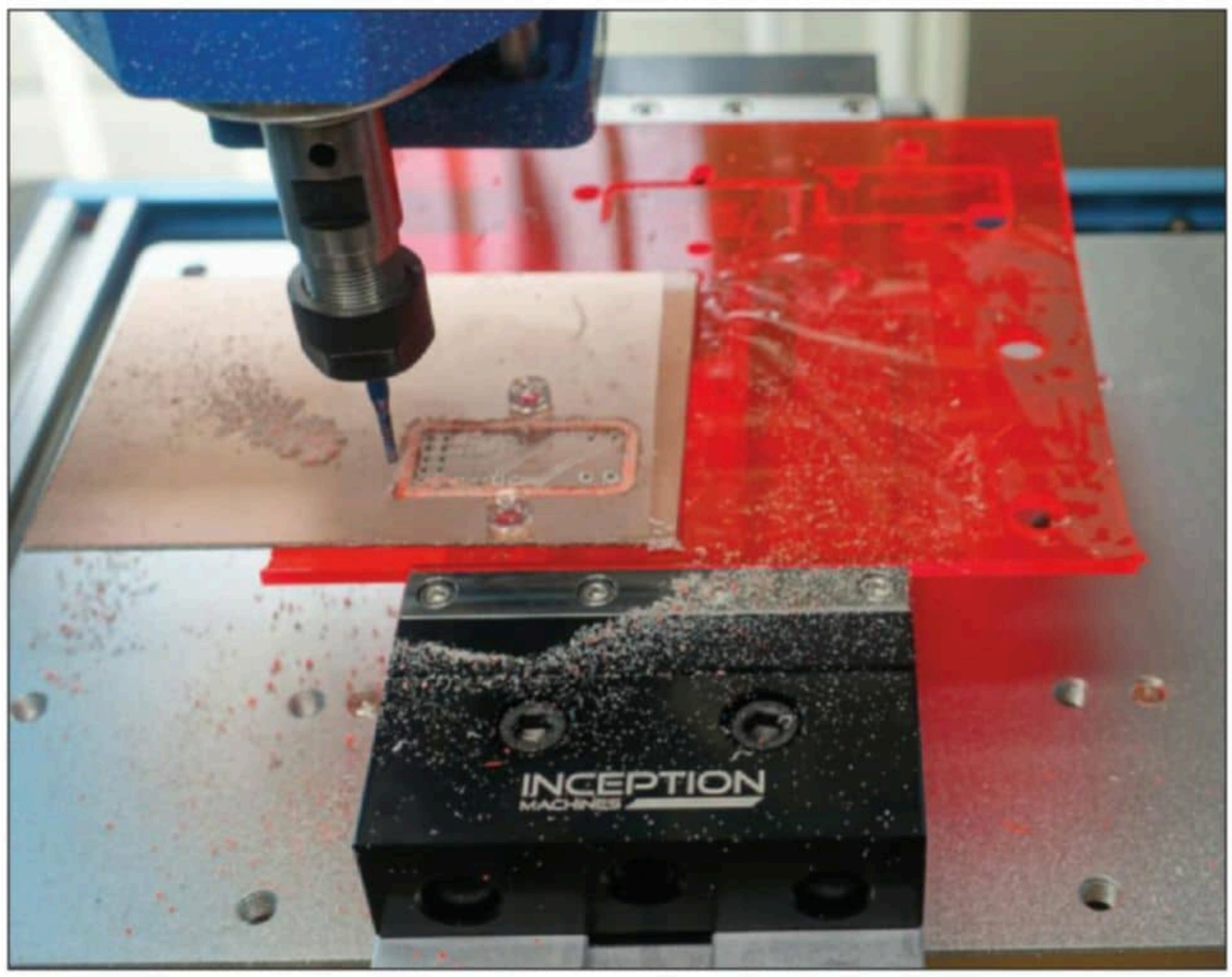


FIGURE 9 PCB after completing milling.

on the resistance setting, you know you have the tool tip at the z-axis origin. Click the z axis origin button on Candle. You only need accurate z-axis positioning for the milling stages. For drilling and cutting out the board, lower the tip of the tool using the manual control on your CNC until you feel it touch the PCB.

Important: Once you have set the z-axis origin, use Candle to raise the tool. The first operation during milling, cutting, or drilling can be to move the tip of the tool over your board without raising it, which will leave a long scratch on your board. Guess how I found this out? Using Candle to raise the tool means that the software will move the tool, then lower it to the board surface to start machining.

To change the tool and to reset the z-axis origin without losing the x, y positioning, use Candle's controls to move the spindle somewhere that you can change out the tool.

Use Candle's controls to move the tool tip back over your PCB milling area. While you move the spindle using Candle software, the position of the spindle remains known to the software. After changing the tool, move it back to around the center of your PCB milling area, then manually reset the z-axis origin as detailed above.

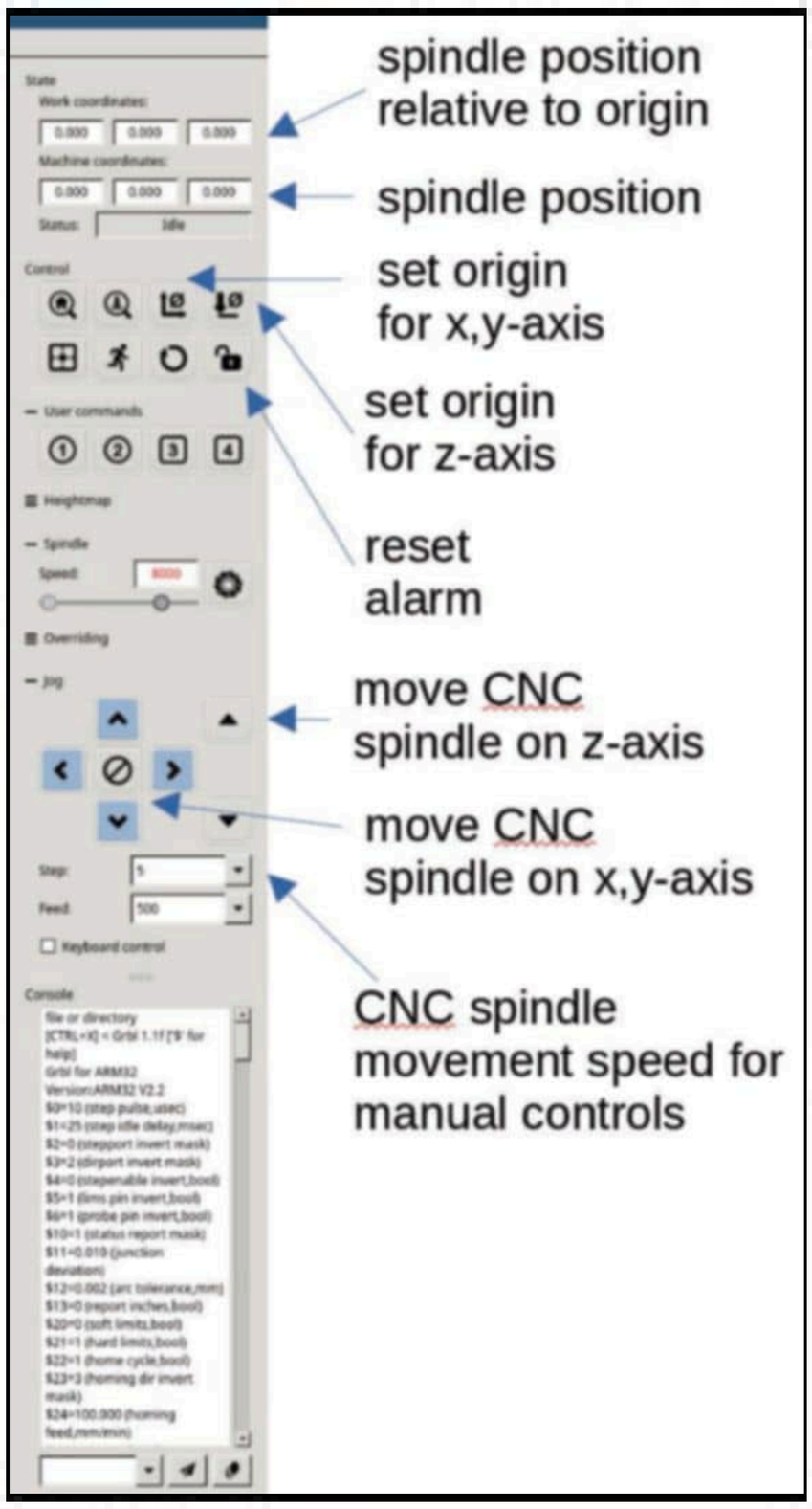

spindle position relative to origin
spindle position
set origin for x,y-axis
set origin for z-axis
reset alarm
move CNC spindle on z-axis
move CNC spindle on x,y-axis
CNC spindle movement speed for manual controls

FIGURE 10
Screenshot of Candle software, with CNC spindle controls annotated.

RECOVERING FROM MISTAKES

If you lose the positioning of your spindle, for instance by manually moving the spindle until one of the end stop alarms is triggered, you can recover. Manually move the spindle until the tip of your tool is just above the center of the bottom drill hole for your alignment bolts. Clear the alarm using the control on the Candle menu. If you use a cross-head bolt, then the center of the cross on the bolt head will be your x,y origin. You can recover the z-axis origin using the same method used each time that you replaced the tool, as discussed above.

Check that milled traces are isolated from the surrounding copper while the PCB is on the CNC, using a multimeter on the continuity setting. If you find continuity, then the milling is not deep enough. You can reset the z-axis origin so that the cutting tool is placed a little more firmly against the PCB, then rerun the milling stage.

If your tool breaks, replace it, reset the z-axis origin and rerun the G-code for that machining step.

LESSONS LEARNED

I made mistakes. It turned out that the contacts on one side of my 3.5mm audio socket disconnected when a 3.5mm plug was inserted. I hadn't accounted for this, so I needed to update the footprint and routing for the socket. I needed to lengthen the board, so that the 3.5mm socket protruded enough to allow for the retaining nut to go over the thread on the socket. This created mechanical stability and reduced the chance of the solder joints on the PCB cracking when a plug was inserted into the socket.

I learned a lot during this project, and have an increased respect for the PCB manufacturers who produce flawless PCBs at a fraction of the price I had to pay when starting out in electronics. I hope that this article is of interest, and that I have encouraged you to give milling your own PCB a go.

ABOUT THE AUTHOR

Matthew Oppenheim works in marine geophysical survey and offshore construction. When not working at sea he loiters at InfoLab21, Lancaster University, where he works on assistive technology projects and how to deploy them into the real world. Visit www.mattoppenheim.com for more details of these projects. Please email matt@mattoppenheim.com with any comments.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
Resources [1] to [13] as marked in the article can be found there.

LILYGO | https://lilygo.cc

30th ANNIVERSARY

DESIGNCON® 2025

WHERE THE CHIP MEETS THE BOARD

JANUARY 28 – 30

SANTA CLARA CONVENTION CENTER
SANTA CLARA, CA

The Nation's Largest Event for Chip, Board & Systems Design Engineers

Created by engineers for engineers, North America's largest chip, board, and systems event, DesignCon 2025, returns to Silicon Valley. This annual event brings together designers, technologists, and innovators from the high-speed communications and semiconductor communities for three jam-packed days of education and activities.

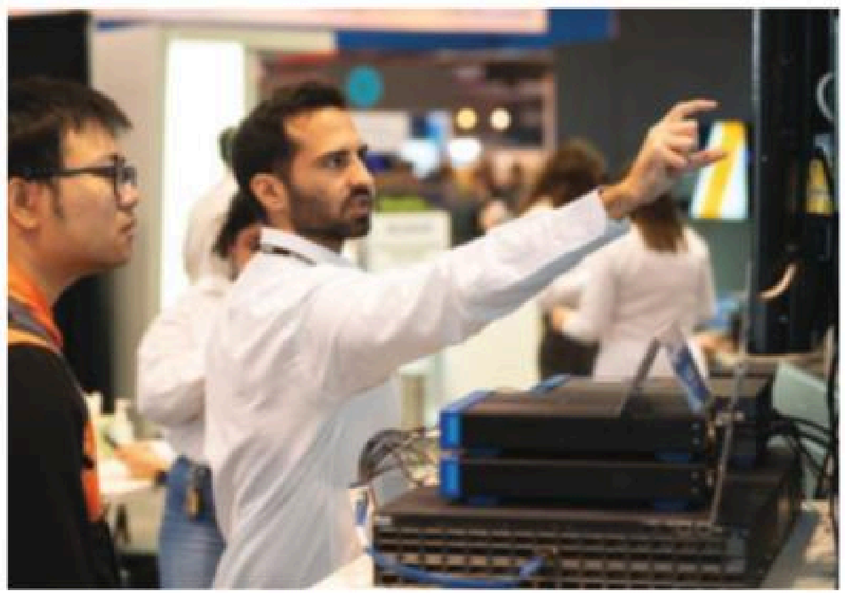
DesignCon is a must-attend opportunity to share ideas, overcome challenges, and source for designs. Join DesignCon at the Santa Clara Convention Center January 28 – 30, 2025!

EDUCATION

- DesignCon's 14 Track Technical Conference
- Drive World Conference Track
- Expert Speakers

EXPO

- Keynote Presentations open to all attendees
- Free Education at Chiphead Theater
- Expo Hall with Leading Suppliers
- Interactive Demos



Save 15% on conference registration or receive a **FREE** expo pass with code **KCK25**.

Register at:

DesignCon.com

informa markets

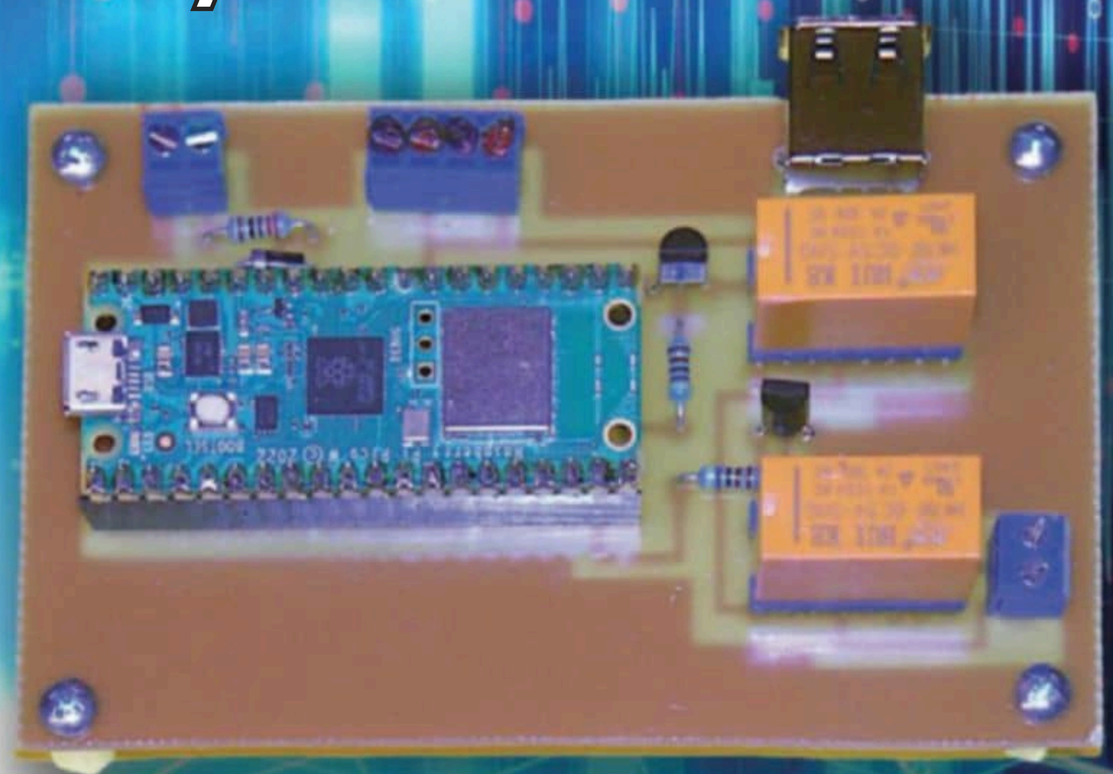
Host Sponsor:

Amphenol

FEATURES

Building a Wi-Fi Router Watchdog

With Raspberry Pi Pico, MicroPython and Thonny IDE



By
Dev Gualtieri

Dev created a watchdog for a Wi-Fi extender using a Raspberry Pi Pico. This monitors Wi-Fi connectivity for his smart home lighting system, which would require a reset twice a year due to rapid power interruptions.

My smart home lighting system has been operational for more than two years. The system uses a supervisory program installed on a Raspberry Pi computer to control the commonly available Kasa smart plugs, wall switches and dimmer modules in a Wi-Fi network. This smart home program on the Raspberry Pi has been robust against power failures, rebooting into the proper state when power returns. Unfortunately, the Wi-Fi extender I use hangs about twice each year, requiring a manual reset.

I had no idea what could cause this problem until a chance occurrence. While working at my desktop computer, I heard a rapid tick-tick from its uninterruptible power supply (UPS). This must have been caused by the power company doing a rapid switching between its circuits. The resulting brief power interruption was too rapid to affect the several digital clocks in my house, including the one at my microwave oven, and it was too rapid to trigger a reboot of the smart home Raspberry Pi. When I checked the status of the smart home Wi-Fi extender, I found that it had hung and did not respond. Now, I had at least one cause for lapses in my smart home Wi-Fi connectivity, and I needed a solution.

Several years ago, I wanted to buy a timer for the exposure station I use for photosensitive printed circuit boards. The customer comments for these devices showed that a common use was a daily power off-on cycle of routers to correct the same problem. This tactic was used by people who need constant access to the video cameras at their remote vacation homes. While effective, this is not an elegant solution, and it would potentially result in a long outage if the router hung just after the power cycle.

If the momentary power interruption I observed was the only reason why my Wi-Fi extender hung, connecting it to a UPS would be one solution. Aside from the cost of such units, there's also the maintenance issue of battery replacement every few years. I'm very sensitive to the growing problem of consumer electronic waste. Recycling of used lead-acid batteries is necessary, but it's an inconvenience.

I found that a newer version of my Wi-Fi extender has a watchdog option that pings a user selected IP address at a selected

interval and reboots the device if the ping is unsuccessful. This is probably the best option for people who have an application similar to my smart home system and would rather buy than build. However, there are cases in which a router replacement can't be done, as when the router is supplied by your internet service provider. I decided against this easy solution of buying the newer model of my Wi-Fi extender, since I had the more grandiose plan of installing my Raspberry Pi smart home server in the same box as a Wi-Fi watchdog. My project box, as shown in **Figure 1**, incorporates the Raspberry Pi 3B+ that I use as the smart home master along with a Raspberry Pi Pico W as the watchdog. I've never had a problem with this Raspberry Pi 3B+, but I decided to add a little more code to make a watchdog for that, also.

The Raspberry Pi Pico W that I used as a watchdog for the Wi-Fi extender in my smart home system is an inexpensive Wi-Fi capable microcontroller that can be programmed using the microPython programming language. Unlike a Raspberry Pi, which is a functional Linux computer, the Raspberry Pi Pico is a microcontroller board similar to an Arduino. The Arduino was previously programmable only using C++, and learning this language is a large hurdle for novices to overcome compared with Python. The Arduino can now be programmed in microPython, and I'm sure that Arduino fans will have no trouble porting my software to an Arduino. Programmers proficient in C++ can use that language for the Pico as well.

FIGURE 1
Interior of the Wi-Fi watchdog, showing the Raspberry Pi Pico W and its associated circuitry on the left, and the Raspberry Pi 3B+ that I use as the server for my smart home application. The Raspberry Pi 3B+ is not required for the Wi-Fi watchdog.

CIRCUITRY

The circuitry, as shown in **Figure 2** is quite simple. Voltage is supplied by a small wall transformer that was repurposed to become an internal power supply by connecting the the AC plug end to an AC receptacle. The 5V power is supplied to the Raspberry Pi Pico through a 1N5819 diode, a Schottky diode, as recommended in the Pico documentation. This diode allows simultaneous connection of the Pico to a host computer, which also provides a five-volt power source through its USB connector, and the internal power supply. Although I haven't tried it, substitution of a conventional diode such as a 1N4004, should also work, but with the effect that the Pico would be powered by the host computer, not the circuit power, when its USB cable is connected.

A top view of the circuit board layout is shown as **Figure 3**. The relays are commonly available devices that can be plugged into DIP integrated circuit sockets. All these relays seem to have the same pin connections, but I found one example for which the coil polarity was important. This was likely a consequence of the relay having an internal permanent magnet that allowed a strong actuation at lower current. I designed the circuit board to accommodate both polarized and non-polarized relay coils.

SOFTWARE

Python has become a popular programming language, principally because of its ability to import code libraries for specific functions. MicroPython, which is Python for microcontrollers, is used by the Raspberry Pi Pico. The first step to programming the Raspberry Pi Pico is to install an integrated

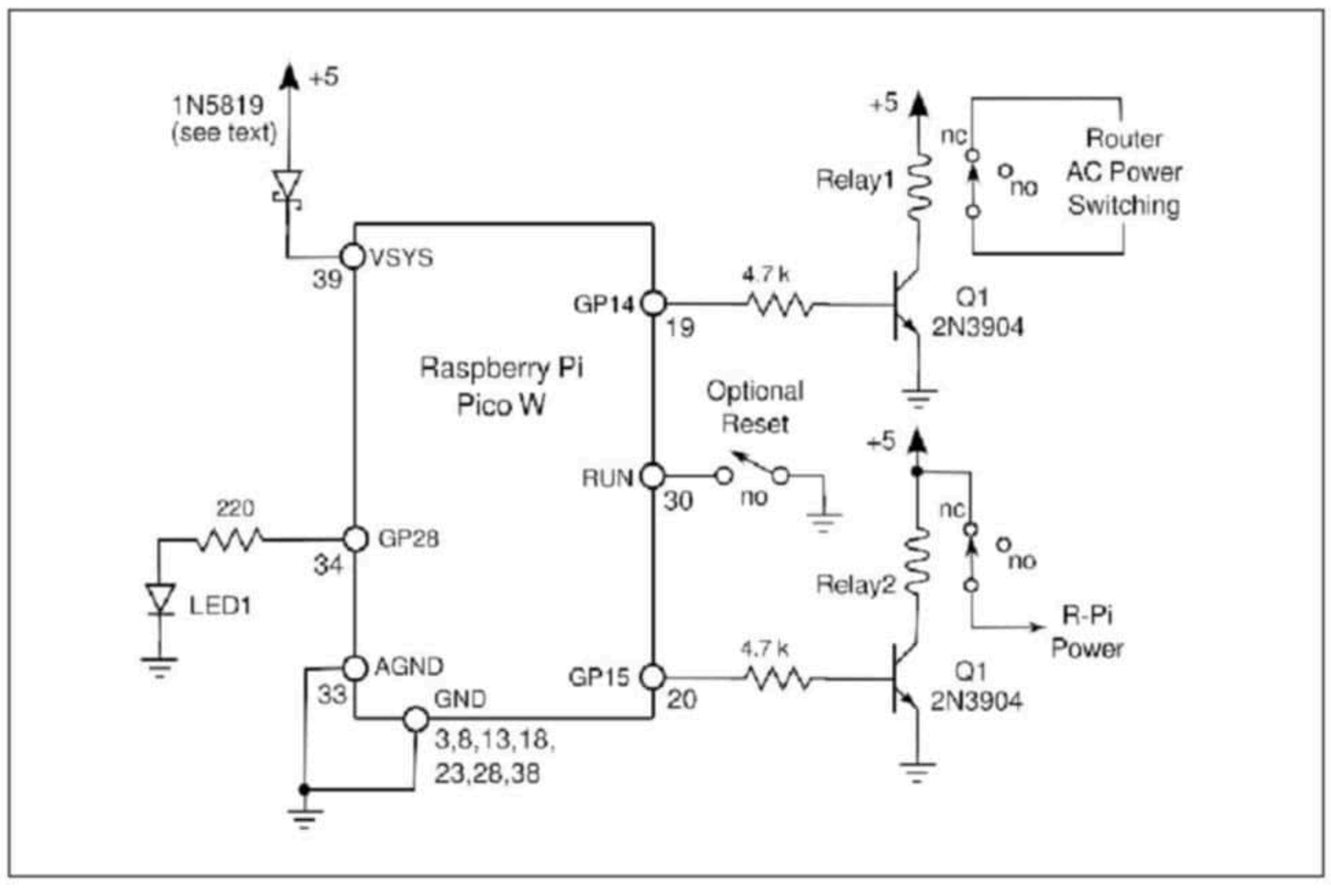


FIGURE 2
Schematic diagram of the Wi-Fi watchdog circuit. If the watchdog is not used to monitor an internal server, as in my smart home application, Relay2 and its associated circuity can be eliminated.

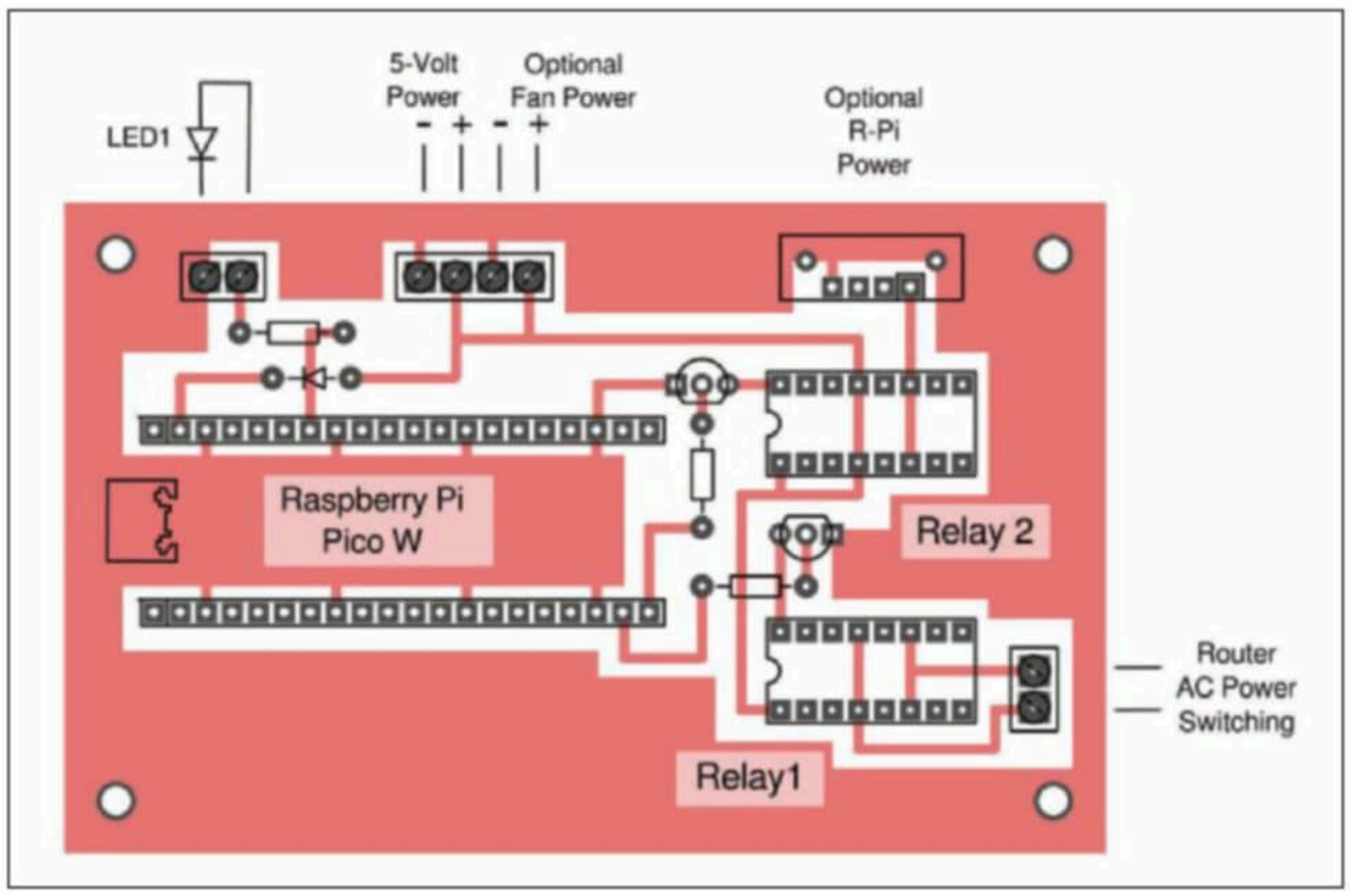


FIGURE 3
Top view of the printed circuit board layout. Note that AC line voltage appears around Relay1. The circuit board conductors should be isolated by an insulating tape for safety.

development environment (IDE) for Python on your host computer. The best choice for this is Thonny, which has versions for Linux, Windows, and MacOS.

Holding the BOOTSEL button on the Raspberry Pi Pico while plugging its cable into a USB port on the host computer establishes a communications link that allows installation of the MicroPython firmware. After that, this button is essentially useless and does not act as a button to reset your application. The Raspberry Pi Pico W will appear as a mass storage device named RPI-RP2 in your file manager. While Thonny on my Linux computer has a menu option to install MicroPython for a Raspberry Pi Pico, this version will only install firmware for the non-wireless version of the Pico. That's fine if that's all you need, but our watchdog program needs Wi-Fi. What you need to do is download the Pico W firmware file, a file with a *. uf2 extension, to your host computer from the Raspberry Pi website [1].

ABOUT THE AUTHOR

Dev Gualtieri received his Ph.D. in Solid State Science and Technology from Syracuse University in 1974. He had a 30-year career in research and technology at a major aerospace company and is now retired. Dr. Gualtieri has a science and technology blog at http://www.tikalon.com. He is the author of three science fiction novels, two science-themed novels for children, and other books about science and math. See http://www.tikalonpress.com for more information.

Additional materials from the author are available at: www.circuitcellar.com/article-materials

References [1] as marked in the article can be found there.

Resources

Raspberry Pi | www.raspberrypi.com

You then copy the *.uf2 to the RPI-RP2 mass storage device. The Raspberry Pi Pico W will reboot, and you are now running the proper version of MicroPython. Thonny has an integrated terminal that allows executing commands, such as the help ("modules") command, which lists all the MicroPython modules available for import. A proper firmware install will list network as one of these modules. Test programs can be saved on the Raspberry Pi Pico W and run from Thonny. Any program named main.py will be automatically executed at each power-up.

The Wi-Fi watchdog program for the Raspberry Pi Pico, named main.py, can be found with other resources on the Circuit Cellar website. When this is installed on the Pico, the following happens at each power cycling or reboot:

- Required libraries are imported.
- GPIO pins 14, 15, and 28 are configured as output pins, and their initial values are set at zero. Pin 28 is the LED pin, pin 14 is the AC power relay pin, and pin 15 is the DC power relay pin.
- A short time delay is done before enabling the watchdog timer. This aids in development using the Thonny IDE, since it allows some file management before the remainder of the program is started.
- The Pico watchdog timer is enabled with an 8 second timeout. This timeout value is near the maximum possible value.
- The watchdog is initially fed, and repeats at intervals of less than eight seconds timeout throughout the program.
- A 45 second time delay is done to allow devices to recover from a power failure.
- The SSID and password of your Wi-Fi router is defined.
- The URL of a check file on the optional integrated server board is defined.
- Functions are created to flash the LED, halt with the LED off, cycle the AC power relay, and cycle the DC power relay.
- A function is defined to check connectivity to the Wi-Fi router.
- An infinite loop is entered to twice check the Wi-Fi router connectivity and cycle its AC power if connection is not established. If connection is not established after that, the LED is set to off to signal a major fault, and the program is halted.

If Wi-Fi connection is established, there's an optional check at to whether a file exists on an integrated server board. If the file is not found, the DC power to the server board is cycled. If another attempt to find the check file fails, then then the LED is set to off to signal a major fault, and the program is halted.

The usual method of checking connectivity to a server is by using the ping command. However, ping is not a native function in microPython, and a ping implementation is an entire page of code. That's why my connectivity check uses the simpler approach of checking access to a server file using a get request. The file I used is a small text file, but a blank file would work as well. **Figure 4** shows the program executing in the Thonny terminal.

MICROPYTHON MODULES

A Wi-Fi router watchdog will only be effective if the watchdog circuit itself doesn't hang. Fortunately, the Raspberry Pi Pico has a hardware watchdog that's activated by a microPython command when WDT is imported from the machine module. One minor problem is that the Pico watchdog will remain active until the Pico is power cycled or reset. If main.py exists on the Pico, a reset will again execute this program, once again activating the Pico watchdog. I've added a slight delay before watchdog activation, and this gives time to do some file management in Thonny before it main.py continues. When doing program development in the Thonny IDE, it's best to use a different name for the program and rename it main.py at the end.

The Wi-Fi watchdog program operates in an infinite loop, and when I first started development, the program would crash with a system error after successfully completing the first few checks. I decided that this was caused by the Pico running out of memory having been bloated by allocated data that was no longer referenced. Adding garbage collection after each cycle solved this problem. By importing the gc module, I used the gc.collect() function at the end of each loop.

TESTING

I've included print statements throughout the microPython watchdog program to indicate its status. These appear when the program is run in the Thonny terminal, but they also appear at the Pico connector when the program is run autonomously as main.py. These can be read by a serial communications program through a USB cable. In my case, I use the Linux communications program, Cutecom, selecting /dev/tty/ACM0 as the serial port with 8 data bits, two stop bits, and no parity at a baud rate of 19,200. Monitoring the watchdog circuit this way will affirm its proper operation.

As everyone knows, strange things happen in computing, and my Linux system drops this serial port after a day's operation. Pulling the USB cable and reconnecting brings the serial port back to life. During all this, the Pico watchdog circuit still functions; so, continuous operation of the serial port isn't required, and I decided against an attempt to find the source of this problem.

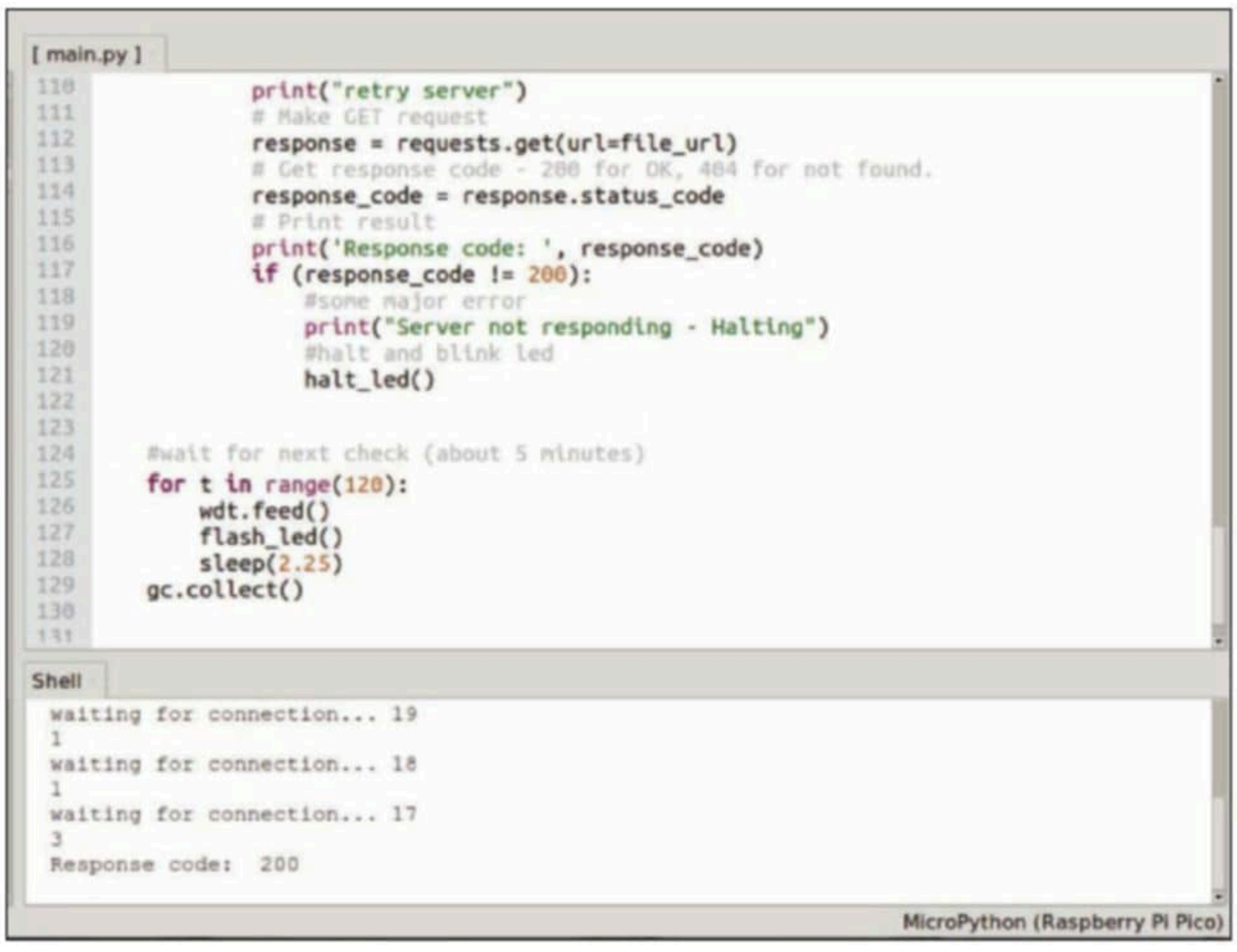


FIGURE 4
A screenshot shows the Wi-Fi watchdog program running in a terminal in the Thonny microPython IDE on a Linux desktop.

RASPBERRY PI PICO TIPS

Unless you order a Pico with pre-soldered pins, expect to solder 40 connections before you start. One confusion that even I faced when first programming the Raspberry Pi Pico is that the pin numbers in microPython are the GPIO pin numbers and not the physical pins of the circuit board.

The connector on the Pico board is a micro-USB connector, and most Raspberry Pi Pico purchases include a micro-USB to a male USB A cable for connection to computers for power and programming. It's common to reboot the Pico by disconnecting and reconnecting the cable at the micro-USB connector, but it's been noted by many that this connector is not mechanically stable, and it may detach from the circuit board after repeated insertions and removals. An alternative reset can be done by temporarily grounding pin 30, the pin designated as RUN.

Inexpensive breakout boards are available in which the Raspberry Pi Pico pins are connected to screw terminals. As one who has purchased such screw connectors for my own boards, I'm amazed at the low price of such boards containing as many as 40 screw connectors. The breakout boards are good for experimentation before you go to your final circuit board build.

A Raspberry Pi Pico W with no power drain from its GPIO pins requires about 150 mA at startup, and about 50mA in its quiescent state. A GPIO pin can source or sink about 15mA, but the total current from all GPIO pins should not exceed about 50mA. A computer's USB 2 connector can supply 500mA, and a USB 3 connector can supply 900mA; so, there's no problem powering these from a computer when drawing current from multiple GPIO pins.

FEATURES

Basic Pulse Circuits

Part 2: Flip-Flops, Timing Parameters and Synchronization

By
Wolfgang Matthes

In part one of a three-part series, Wolfgang wrote how basic pulse circuits help digital circuits, such as embedded boards with ARM processors, deal with pulse trains or bursts of pulses from the outside. In Part 2, he dives into enabling flip-flops, timing parameters, and synchronization, design tasks needed to capture, detect, and filter pulses.

Many digital systems must process signals from outside from sensors and level translators. The signals are mostly pulse trains or bursts of pulses. Some pulses may be too narrow, some may be too wide, and other pulse trains may be affected by glitches. Designers need to learn about capturing pulses, detecting edges, filtering out or detecting glitches, and synchronizing the pulses with internal clocks. In the first part of this three-part article, we introduced the latches and flip-flops as the most basic components to address these fundamental design problems. This month, we examine how to enable and disable flip-flops, timing parameters and synchronization. These are minor but intricate problems most textbooks do not cover. Solving such problems requires some familiarity with the appropriate components and some inclination to deal with the utmost details.

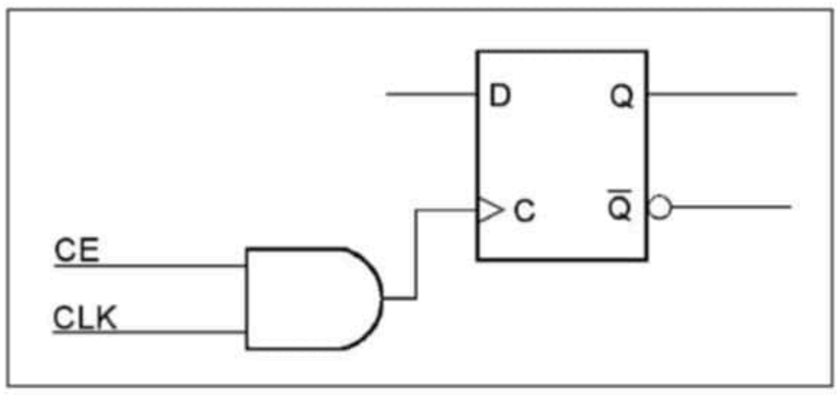


FIGURE 22
This straightforward implementation of the clock enable function will not always work. It's better not to try it.

ENABLING AND DISABLING FLIP-FLOPS

An enabled flip-flop will react to an appropriate clock edge precisely as commanded by its input signals. A disabled flip-flop disregards the input signals. It keeps its state and does nothing else. Whether the flip-flop is enabled or disabled is controlled by a clock enable (CE) signal. We could think of implementing this function by simply turning the flip-flop's clock off if the clock enable signal (CE) is inactive (**Figure 22**). The principal drawback is that the clock enable signal must be, as long as the clock is active, as clean as the clock itself; it is not allowed to behave like other clock-synchronous input signals (as shown later in the article). Hence, the preferred operation is to let CE enable or disable the flip-flop operation while the clock runs continuously. The flip-flops in the

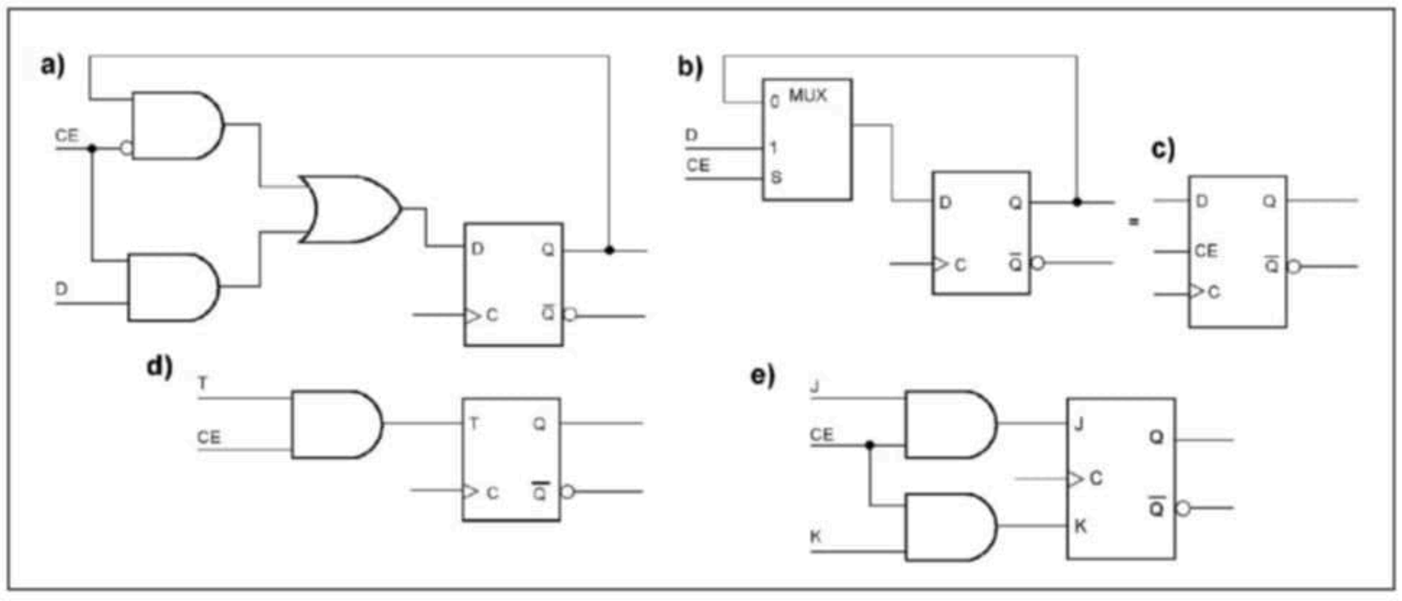


FIGURE 23
Preferable implementations. DE flip-flops (that is, D flip-flops with clock enable) may be implemented with gates (a) or a multiplexer (b). If CE = 0, the flip-flop's state is held via the feedback that is then enabled (a) or selected (b). (c) shows the symbol of the complete DE flip-flop; (d) and (e) show how the enable function may be added to T and JK flip-flops.

cells of Complex Programmable Logic Devices (CPLDs) and FPGAs typically have the CE function built into the hard-wired flip-flop circuit. It will be invoked by programming, so you don't need to use up combinational resources like product terms or lookup tables to implement it for yourself. Otherwise, the clock enable function is to be implemented by combinational circuits upstream of the flip-flop's synchronous inputs (**Figure 23**). When you enter your design ideas by behavioral description (for example, in Verilog), you don't need to bother enabling the flip-flops. The compiler will do that for you.

DESIRED BEHAVIOR

By combinational circuitry and feeding back the output, we may implement each synchronous behavior including SET and RESET by each flip-flop type. Designing with D flip-flops is easiest because we need only to think up the desired transfer function and implement it with gates (or let the IDE do the clerical work). The **Figures 24-28** show some examples. When the flip-flops need an enable (CE) input, swap the elementary D flip-flop against a DE flip-flop.

Typical problems are related to signal edges not steep enough, pulses not wide enough, and signal traces or wires that may propagate noise.

Because they rely on feedback operation, latches and flip-flops are much more susceptible than combinational logic. The input signals of the latches and flip-flops should fulfill the static and dynamic requirements (that is, levels, widths, and rise and fall times) as stipulated in the datasheet. Occasionally, it may be necessary to insert level translation or Schmitt trigger circuits in the signal paths (or to program the I/O cells appropriately). The rules and recommendations concerning bypass capacitors, the routing of signal and power supply traces, grounding, and so on should be heeded carefully.

Noise coupled in via the output wiring could easily make the circuit change its state.

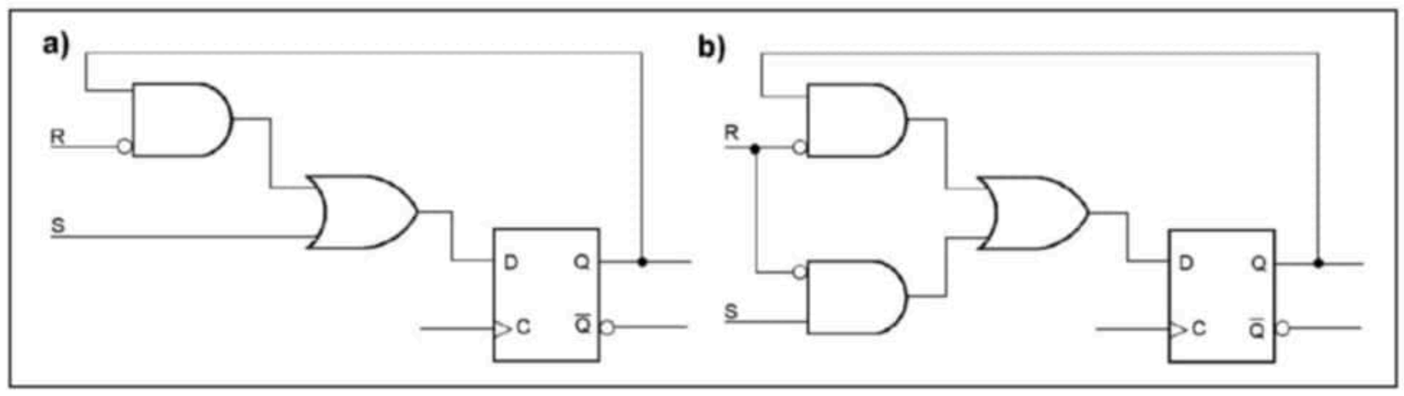


FIGURE 24
Synchronous RS flip-flops. The "forbidden" input condition R = S =1 is allowed here and implemented according to a rule of priority or dominance. In circuit (a), the OR gate propagates a logical one on the S input to the flip-flop's D input. Hence, set (S) dominates over reset (R). In circuit (b), the inverted reset (R) signal inhibits the feedback as well as the propagation of the set (S) input. Hence, reset (R) dominates over set (S).

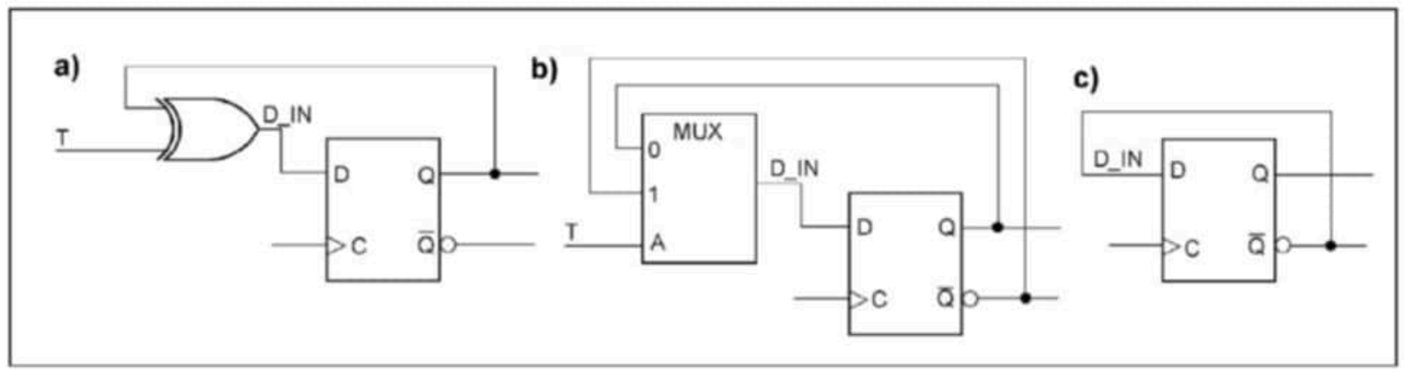


FIGURE 25
Toggle flip-flops. (a) and (b) with, (c) without the T control input. If the D flip-flop provides no inverted output, use an inverter.

FIGURE 26
JK flip-flop. It is similar to the RS flip-flop of Figure 24b, but if J = K = 1, K inhibits the feedback of the flip-flop's output, whereby J gates the inverted output to the data input. Thus, the flip-flop will change its state like a T flip-flop. D_IN refers to Figure 27 and Figure 28.

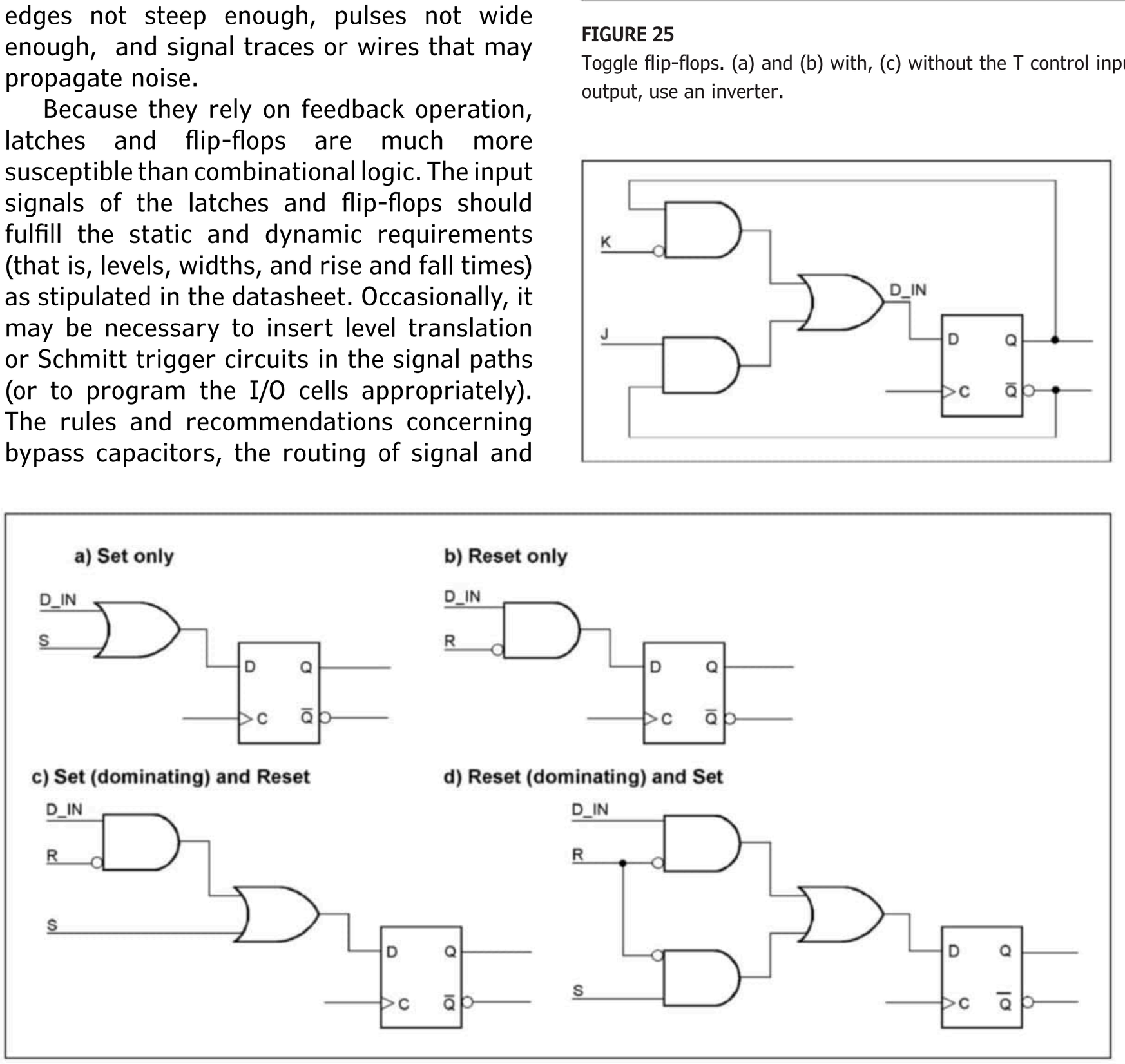
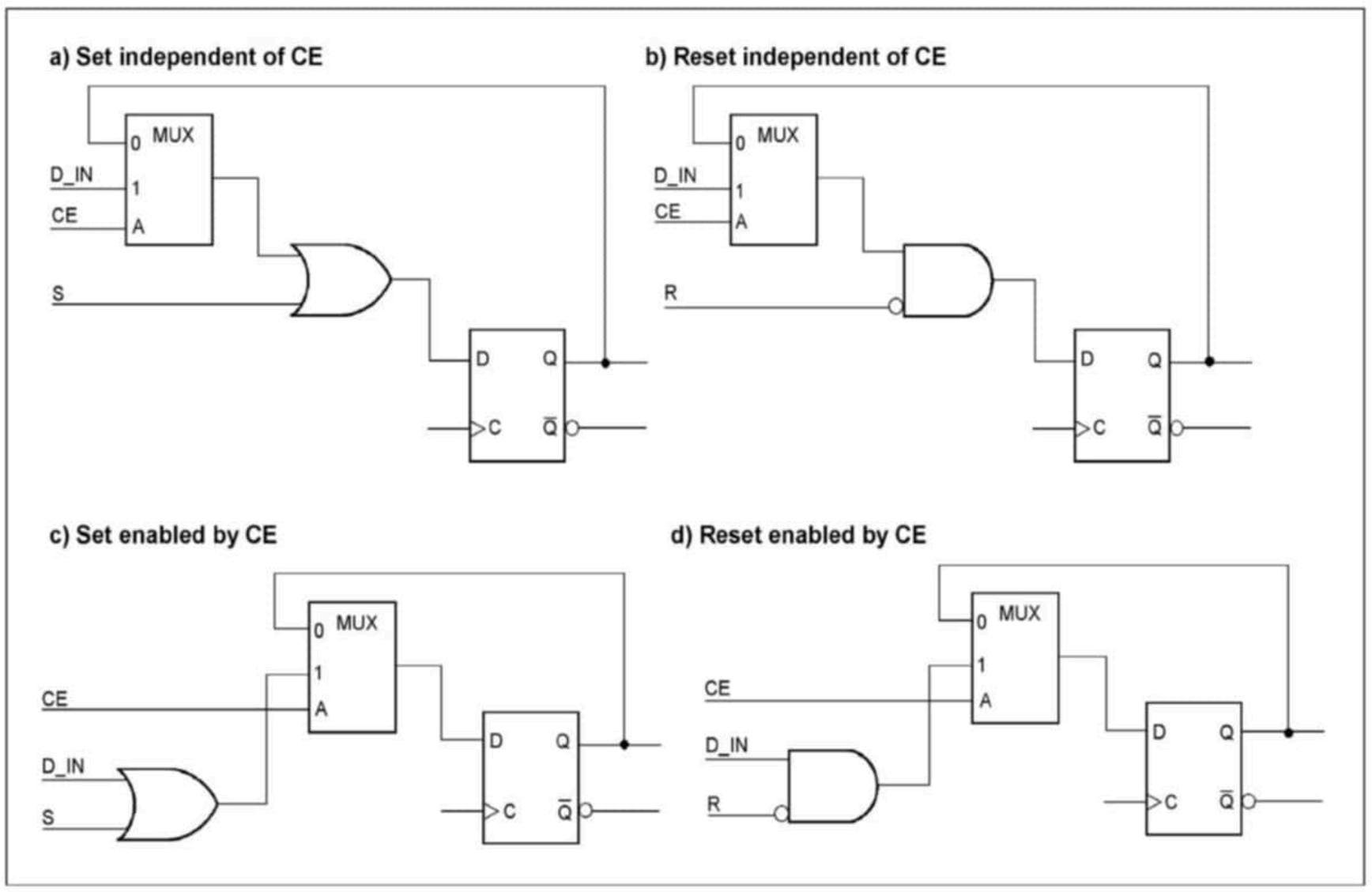


FIGURE 27
Supplementing flip-flops by synchronous set (S) and reset (R) inputs. Such inputs can be added to the flip-flops shown in Figure 25 and Figure 26. For this, cut off the signal line connecting the combinational circuit to the flip-flop's D input and insert one of the circuits shown here. D_IN is the output of the respective combinational circuit.

FIGURE 28
When setting or resetting flip-flops controlled by a clock enable (CE) signal, we must consider whether set or reset should depend on CE or not. When not, set (a) or reset (b) must affect the flip-flop despite an inactive CE. The synchronous set or reset inputs of the CPLD or FPGA flip-flops will be effective regardless of the level of the CE input. When you want the set or reset operations controlled by CE, you must attach the signals via combinational gates as illustrated in (c) or (d).



Assume that the clock is Low and a zero is stored in the slave latch. Now, imagine a positive glitch cross-coupled into the output signal line. It is easy to recognize that it will reach the NAND in the feedback path. Its output will go to zero, thus causing a one at the flip-flop's output.

The principal remedy is buffering. Avoid longer signal lines at the outputs of latches or flip-flops (**Figure 29**). This is especially recommended if the traces on the PCB are long (say, one foot or more) or if the signals leave the PCB. You may not always rely on the CPLD's or FPGA's output buffers. Sometimes, only external buffering will make the circuit work faultlessly. When designing in earnest, consult the appropriate application notes of the CPLD or FPGA manufacturer. Further help you may find on support pages and in user groups. When your design shows intermittent errors, coupled-in noise is a quite likely cause, especially if the Mean Time Between Failures (MTBF) is comparatively short (say, seconds to minutes).

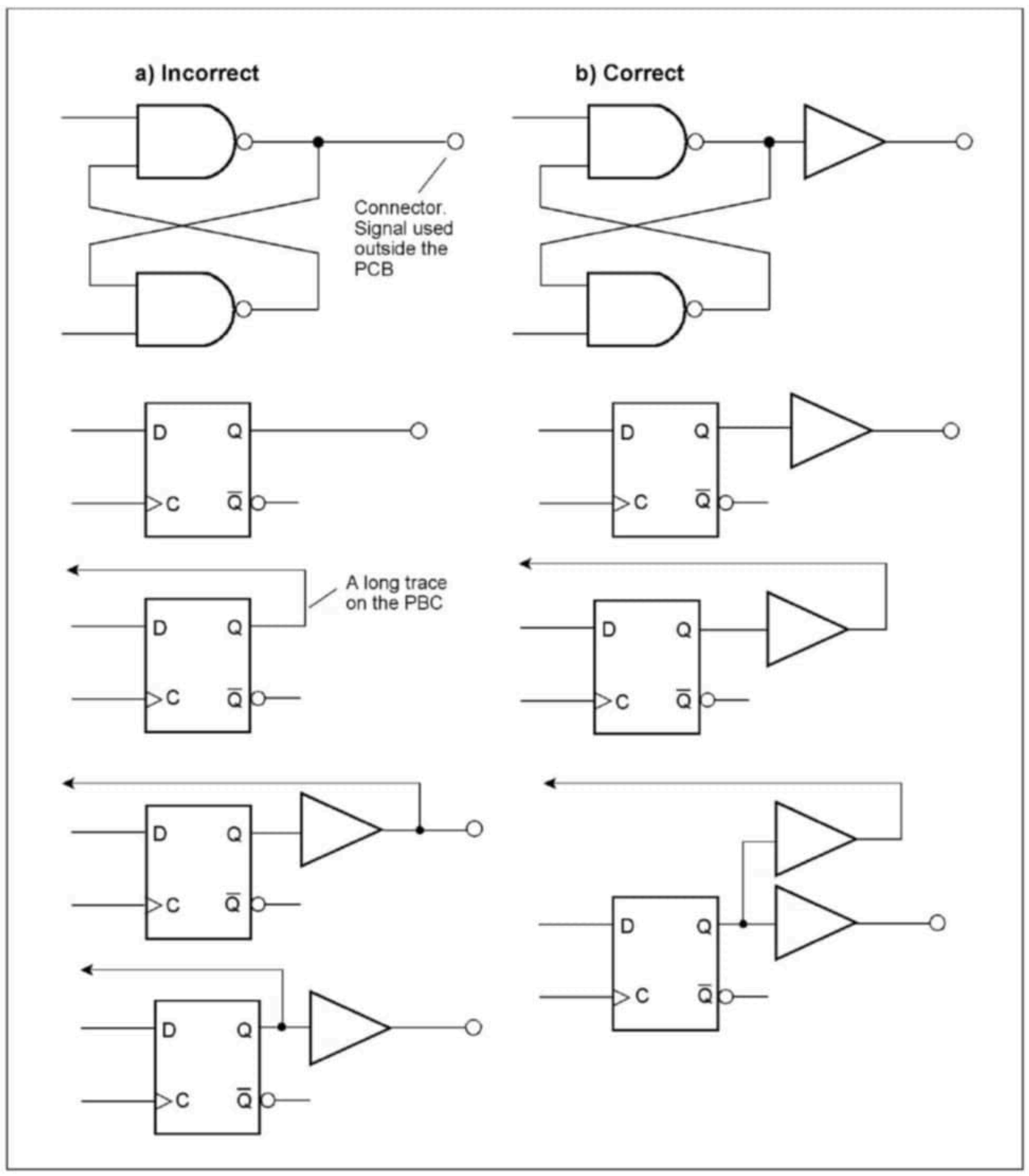


FIGURE 29
Long traces and wires at the outputs of latches and flip-flops. The best connection is point-to-point without stubs or branches. PCB traces and signals routed to connectors should be buffered separately.

TIMING PARAMETERS

In the beginning, the logic family is to be selected. Flip-flops are only a part of it. The static parameters, packages, technologies, and architecture (if we think of CPLDs or FPGAs) we may consider as given. We must deal chiefly with timing parameters like propagation delay times, pulse widths, and timing relationships between signal edges.

RS LATCHES

The pulse width parameter tw specifies the minimum duration of a set or reset pulse

(**Figure 30**). tPDLH is the typical or maximum propagation delay between the leading edge of a set pulse and the Q output going High. tPDHL is the typical or maximum propagation delay between the leading edge of a reset pulse and the Q output going Low.

For an SSI IC or a programmable function in a logic cell, the values are taken from the datasheet. For a makeshift circuit, we must find out ourselves. The S or R pulse should be held active until the output of the first gate has switched. Thus, the pulse width tW should be at least one gate propagation delay. With somewhat more than twice this value, we will be on the safe side. If, as shown in Figure 30, the latch is made of two gates, tPDLH corresponds to one gate propagation delay, and tPDHL two such delays. (Remember, however, not to forget decoupling if the output signal trace does not end in the near vicinity.)

SYNCHRONOUS OPERATION

The timing parameters relate to the clock, the synchronous inputs, the output, and the relationship between changing levels at the inputs and the clock edges. Our example is the D flip-flop. The parameters are, however, valid for other flip-flop types as well.

CLOCK AND DATA SETUP-HOLD

There are two datasheet values concerning an interval in which the data must have settled: the setup time and the hold time (**Figure 31**). The setup time tSU is the minimum interval the input level must be valid and settled before the clock edge triggers the flip-flop. The hold time tH is the minimum interval the input level must be kept valid after the clock edge has occurred. Many flip-flops have a hold time of zero specified. Then, the input signal may change once the clock edge has passed the threshold voltage.

A significant advantage of synchronous operation is that the signals at the synchronous data or control inputs must only be valid and settled in the setup-hold interval surrounding the clock edge (**Figure 32**). Races, hazards, and other glitches do not harm provided they fade away early enough.

PULSE PARAMETERS

The pulse width parameter tw specifies the minimum width of the clock pulse. tPDLH is the typical or maximum propagation delay between the valid clock edge and the Q output going High. tPDHL is the typical or maximum propagation delay between the valid clock edge and the Q output going Low (**Figure 33**). Typically, both propagation delays may be assumed equal. Occasionally, this parameter is called clock-to-output delay tCO.

MAX CLOCK FREQUENCY

Maximum frequency or fmax is an appealing datasheet value. We should, however, be aware of what this specification

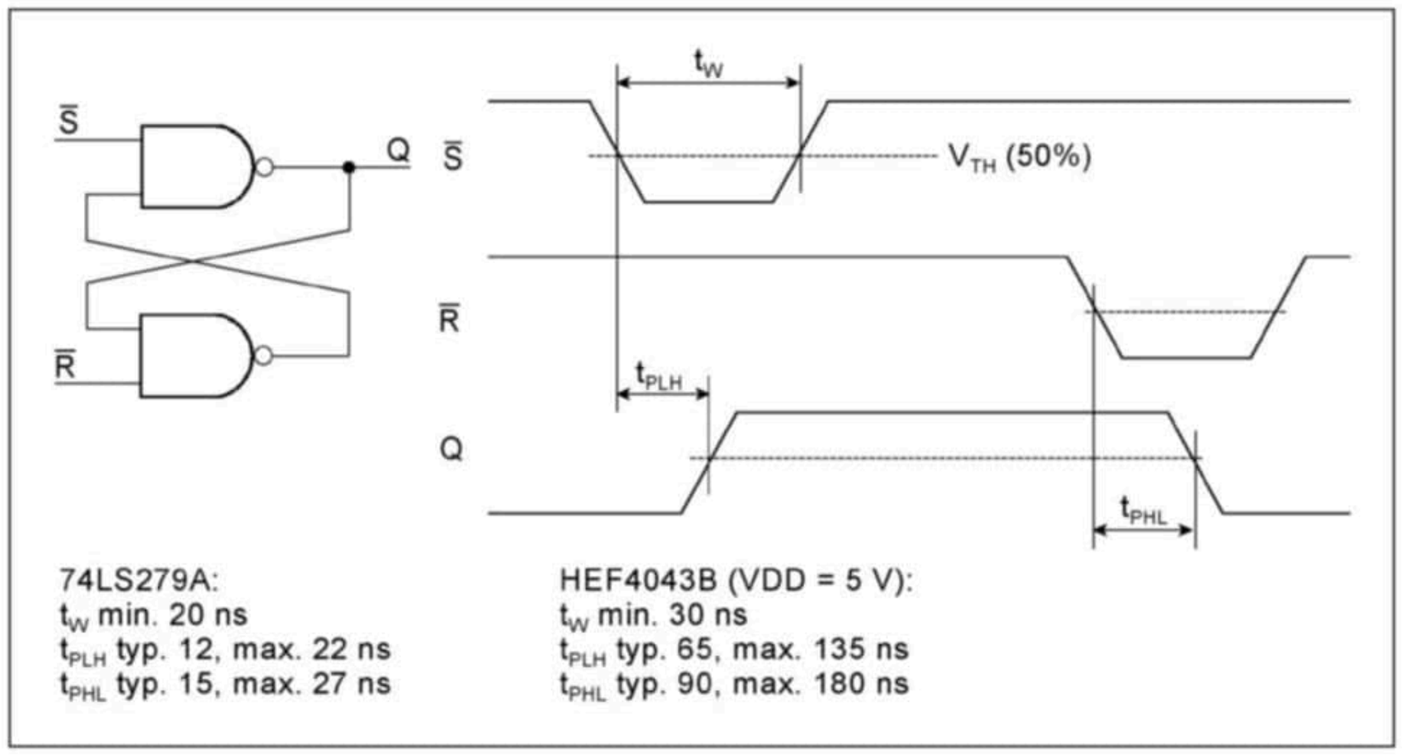


FIGURE 30
Timing parameters of an RS latch.

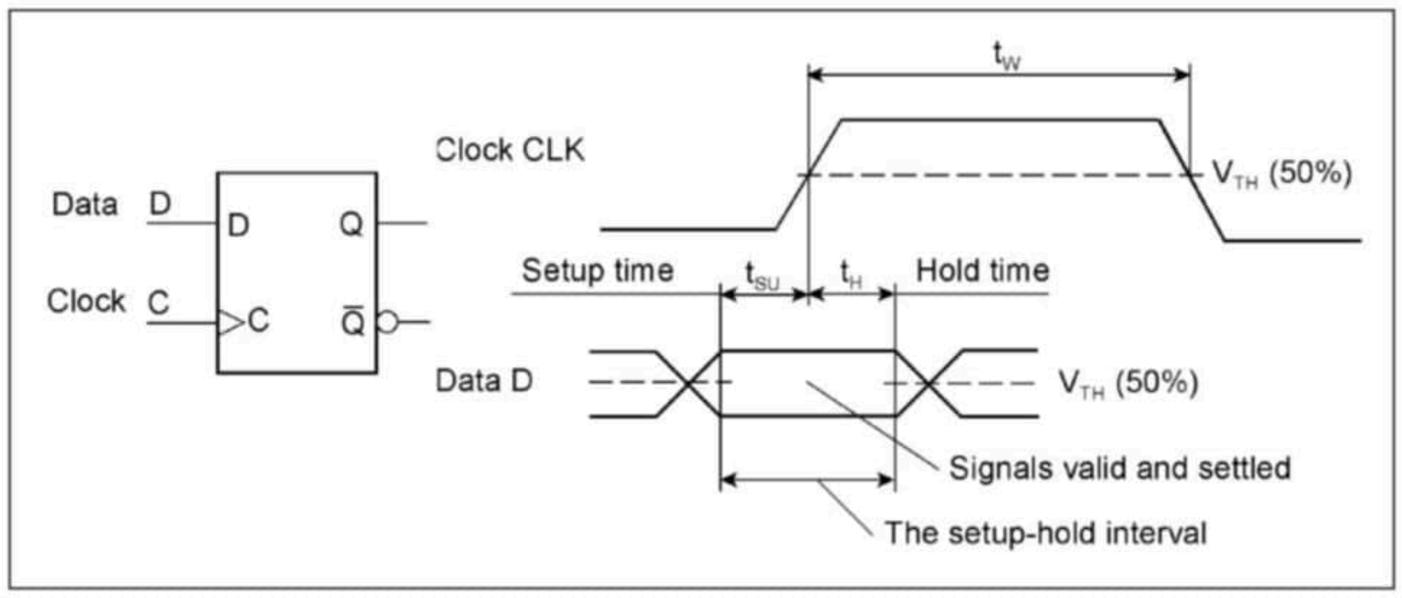


FIGURE 31
The setup-hold interval.

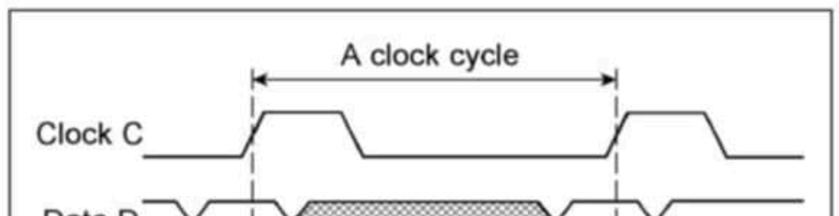

A clock cycle
Clock C
Data D
The setup-hold interval. Data must be valid and settled
Outside the setup-hold interval, data may switch wildly or be affected by coupled-in noise

FIGURE 32
How data signals are allowed to behave.

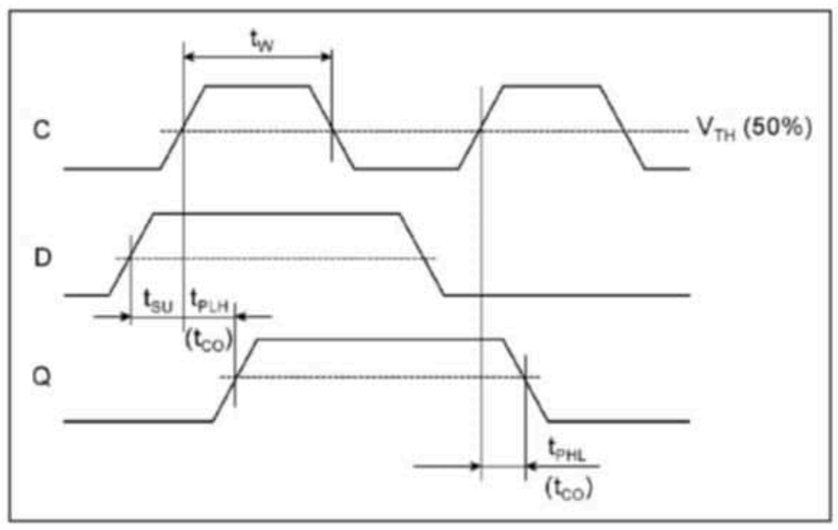
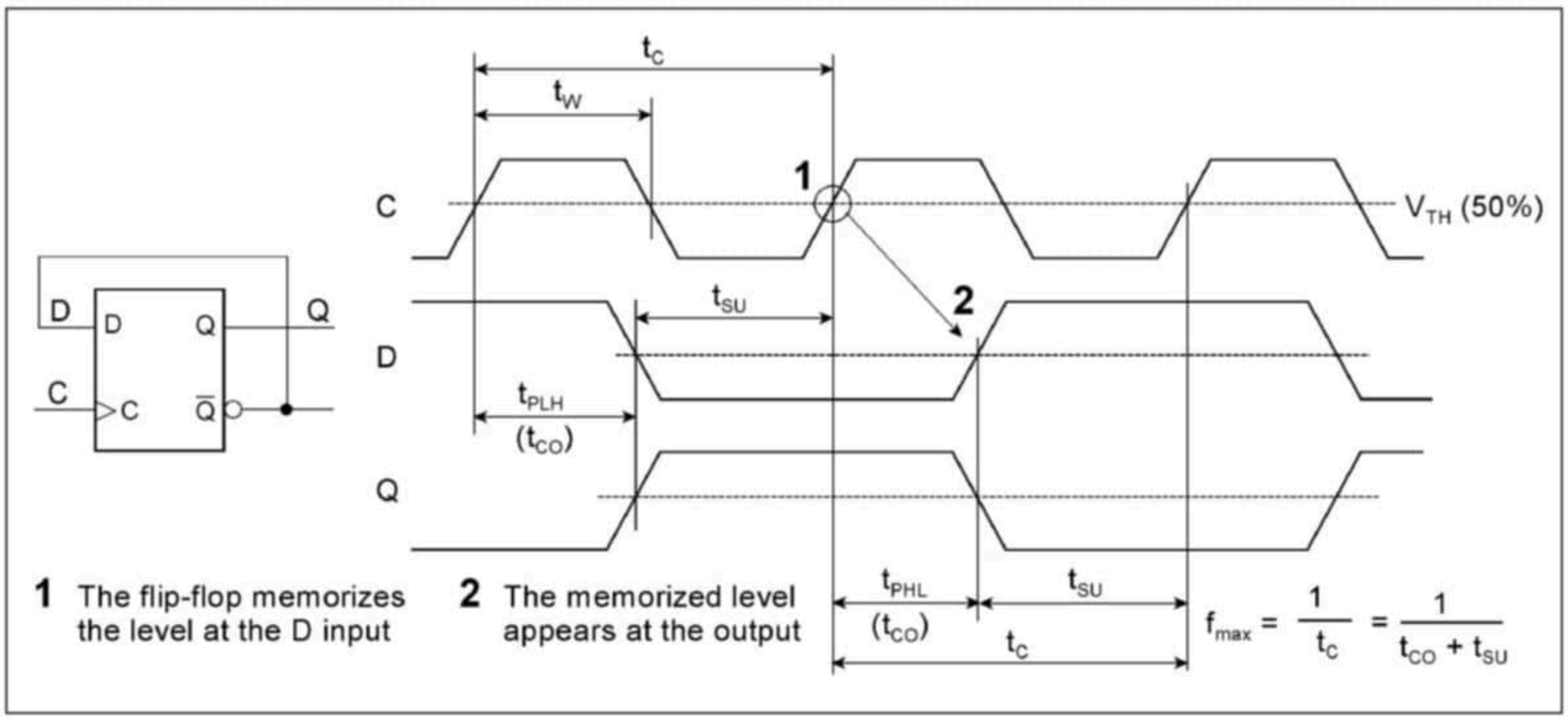


FIGURE 33
Clock and data.

FIGURE 34
Rating the maximum clock frequency by the shortest toggle operation.



really means and how to interpret it. Speaking of SSI and MIS components, like counters and shift registers, fmax is the maximum clock frequency where the component will execute its most basic operation. A 100-MHz counter will count, a 100-MHz shift register will pass bits through, and so on. Nothing is said, for example, how to load it, how to decode the content or to move it to receiving registers, and so on. In a nutshell, when you have to design a 100-MHz circuit, you will be definitely not successful when purchasing 100-MHz rated components.

Flip-flops are rated similarly. Except for dual-edge or dual data rate (DDR) operation, the clock is not required to be a symmetric pulse train. Concerning the minimum pulse width tW, symmetric pulses are assumed. Thus, the minimum clock cycle equals 2 tW, and the reciprocal yields the maximum clock frequency fmax. However, this value is more often than not misleading. With a flip-flop operated under these conditions, you cannot do something with it. Perhaps, it accepts data and guides them, during one clock cycle, to the output, but it cannot toggle, cannot be operated in a shift register, and so on.

A better applicable fmax rating considers the propagation delay from the clock edge to the output tCO and the setup time tSU (**Figure 34**). The most simple flip-flop operation changing its state is the toggle operation. What has to happen in a clock cycle? Following the clock edge, the input level must appear at the output, requiring the propagation delay tCO. Then, the output data must be presented at the input at least as long as the setup time parameter stipulates. Hence, the minimum clock cycle must be as long as tCO + tSU. fmax is the reciprocal.

The toggle operation shown in Figure 34 is only implemented by wire, feeding the inverted output back to the data input. Hence, fmax is the flip-flop's very own speed parameter, not depending on combinational circuitry. In some programmable IC families, the toggle function is implemented in the hard-wired flip-flop circuit and may be invoked by programming. When striving for utmost speed, it could be more appropriate to base the state machine design on T instead of D flip-flops. In most

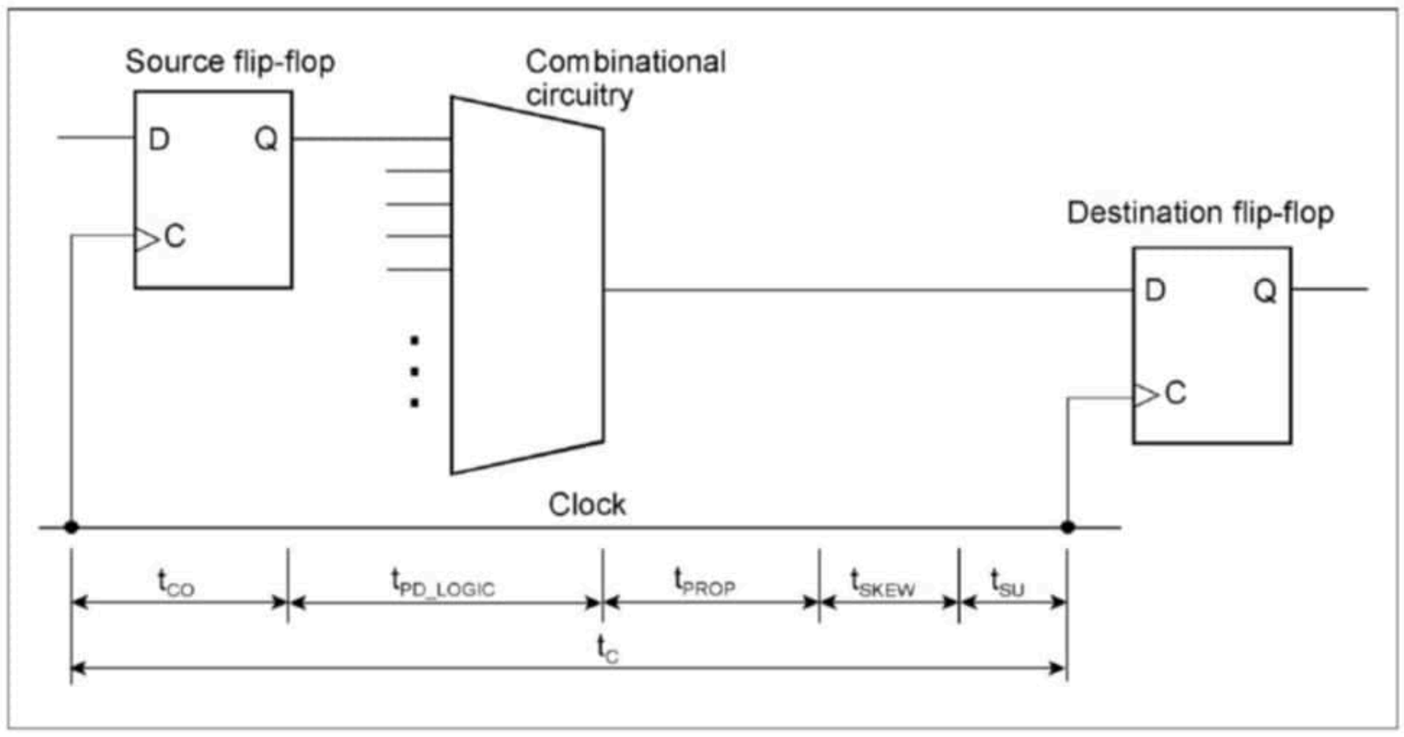


FIGURE 35
Those components add up to the cycle time.

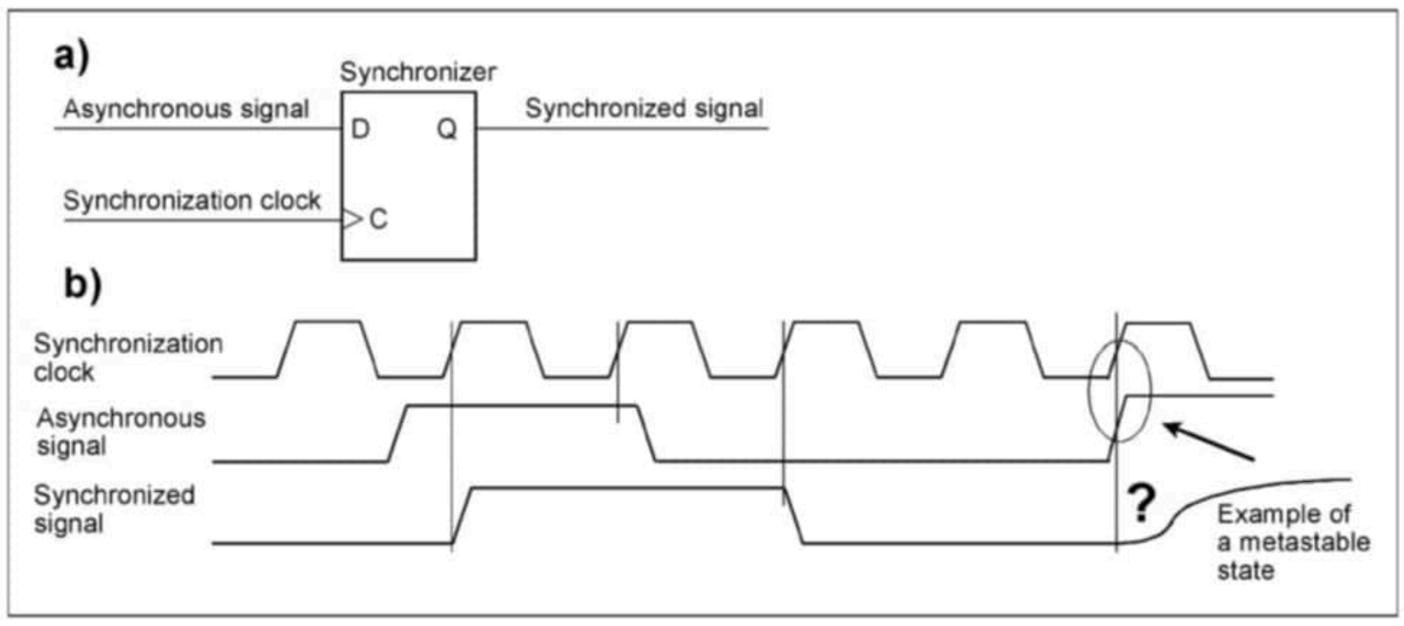


FIGURE 36
Synchronizing asynchronous pulses. a) shows the most straightforward circuit, b) what may happen because the clock edge hits the asynchronous pulse train randomly. Then, by chance, a metastable state may occur, where the flip-flop doesn't output an acceptable logic level, as illustrated by the "?".

circuit designs, however, we cannot get by without combinational circuitry. So, we have to undertake a painstaking timing analysis to find out with which clock frequency our circuit may be run [1] [2].

It's all about additional delays. **Figure 35** servers as an introductory overview. To the source flip-flop's propagation delay (tCO) and the destination flip-flop's setup time (tSU) add up the propagation delay of the combinational (logical) circuitry (tPD_LOGIC) and the interconnections (tPROP) and the tolerance of the clock distribution (tSKEW).

To be able to design circuits that work, you must become familiar with the details of the timing model of your components and, at least when designing for FPGAs, with the timing analysis tools of your integrated development environment (IDE). When designing with tiny logic or other SSI components, look up the component's propagation delay in the datasheets. Notice, however, that the propagation delays depend heavily on the supply voltage (the higher, the lower) and the capacitive load (the higher, the higher). Further, you may have to consider whether to assume typical or extreme (max or min) values. In the distant past, designers were provided with tables or diagrams showing when they were allowed to calculate with typical or statistical values or when they should assume the worst case. Because combinational circuits will not be as deep today, it will be prudent to assume worst-case values. This is because nobody will design SSI circuits where signals flow through 20 gates, one behind the other, so that deviations from typical values may cancel out. When designing for manufacturing in series and cost is to be minimized, you should eventually team up with the quality control people.

The timing model of a CPLD [1] is much more straightforward than that of an FPGA ([3]). When your flip-flops and combinational functions fit in one macrocell or configurable logic block (CLB), it could be sufficient to calculate with the appropriate datasheet values and add a generous allowance. Otherwise, there are no shortcuts, especially if you want your circuit to run with hundreds of MHz. The all-in-one FPGA (containing both application-specific circuits and hard-wired microcontroller cores) is a compelling solution. It requires, however, appropriate development tools and comprehensive experience to make good use of them. Otherwise, the CPLD-microcontroller combo could be a viable alternative (think, for example, of a CPLD containing glitch filtering, and edge detection connected to a PIC, AVR, or ARM microcontroller). There are two packages, but, at least, the hardware design is much easier.

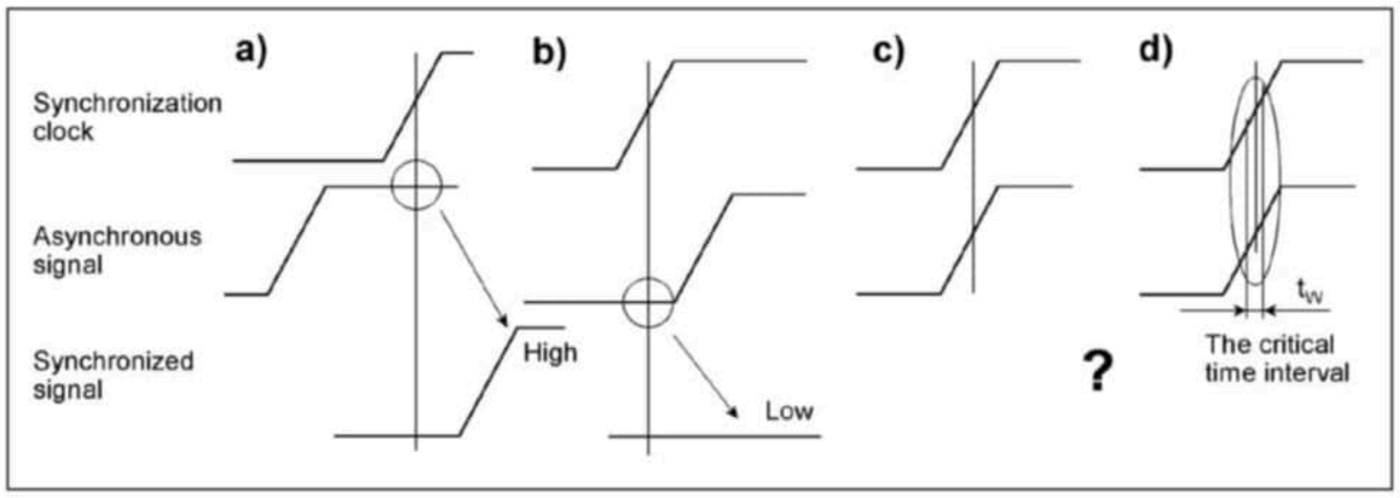


FIGURE 37
The sampling process in more detail. (a) and (b) shows the asynchronous signal being sampled without a hitch. In (c), however, both edges coincide. When this coincidence takes place in a distinct critical time interval, a metastable state may occur (d).

SYNCHRONIZATION

Synchronization means to sample signals coming from outside, making them fit the timing requirements of a clocked circuitry.

In microcontrollers, synchronizers are usually considered a matter of course. In many datasheets and manuals, they are not even mentioned and also not shown in the block diagrams. Nevertheless, you may expect synchronizers built into general-purpose I/O ports and interface adapters. Typical exceptions are external memory and I/O bus systems. If I/O ports are programmed to act as a bus system, the synchronizers are circumvented. The reason is that synchronization would cost at least one additional clock cycle. To boot, it is unnecessary here because bus cycles generated by the microcontroller are already clock synchronous. Only when you want to read signals from outside you need synchronization. Another exception could be edge detection and interrupt triggering. Some microcontroller families let you choose between synchronized and asynchronous operation. The latencies and minimum pulse widths make the difference.

When designing your own circuitry, you will have to solve the problem by yourself. The most straightforward synchronizer is a

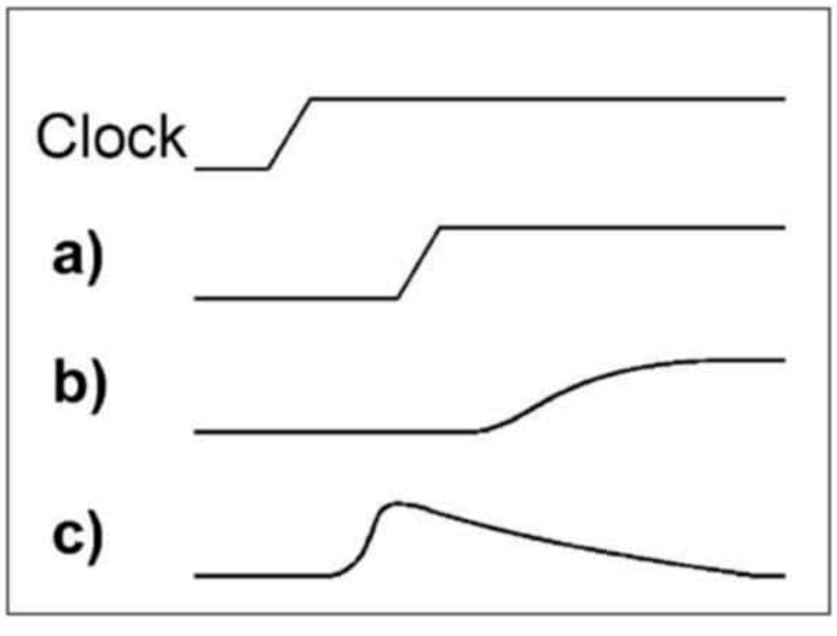
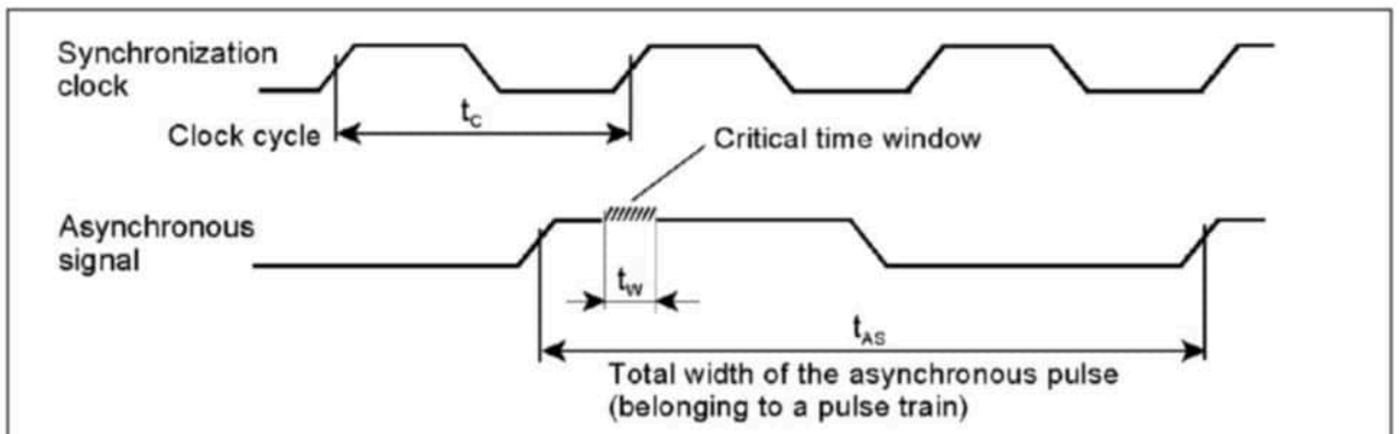


FIGURE 38
Typical output signals of a synchronization flip-flop (patterned after [5], [6]). The line in (a) depicts a correct output signal; (b) and (c) are output signals typical of metastable states. After a while, the metastable state fades away; the signal then enters one of the two logic levels.



A pulse t_{AS} wide can be divided into t_{AS}: t_W critical intervals. The chance to hit one is:

$$1/\frac{t_{AS}}{t_W} = \frac{t_W}{t_{AS}}$$

The chance to fall into a metastable state repeats cyclically according to the clock frequency t_C. Therefore the probability of a hit (that is, of a metastable state) is:

$$\frac{t_W}{t_{AS}}/t_C = \frac{t_W}{t_{AS} \cdot t_C}$$

The reciprocal of the probability* of a metastable state is the mean time between failures $MTBF_{MS}$:

$$MTBF_{MS} = \frac{t_C \cdot t_{AS}}{t_W} = \frac{1}{f_C \cdot f_{AS} \cdot t_W}$$

(The cycle times have been replaced by their reciprocals, the clock frequency f_C and the pulse repetition frequency f_{AS}.)

As an example, let f_C = 2 MHz, f_{AS} = 10 kHz, and t_W = 50 ps. This yields:

$$MTBF_{MS} = \frac{1}{2\ MHz \cdot 10\ kHz \cdot 50\ ps} = 1\ s$$

*: This is only a colloquial term. In exact sense, it is a failure rate.

FIGURE 39
Metastability problems may occur if the signal edge coincides with the clock edge in a narrow critical time interval.

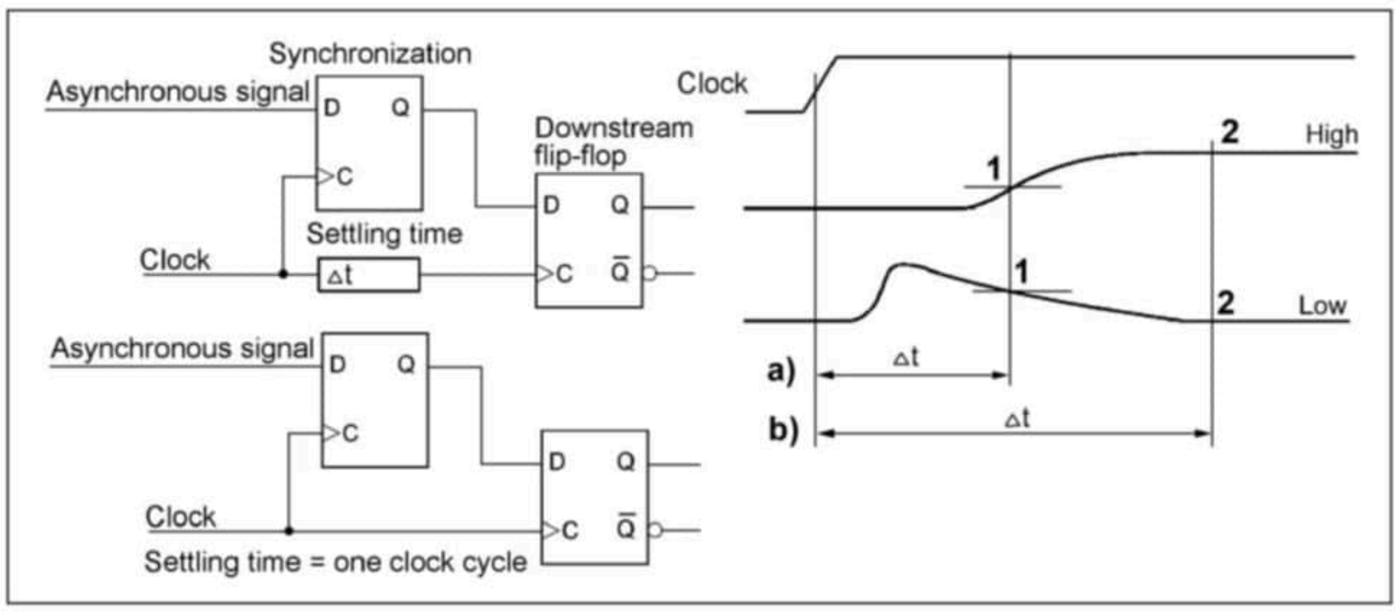


FIGURE 40
Metastable states will settle if we wait long enough. a) The settling time Δt is obviously too short. b) The settling time is long enough; the downstream flip-flop sees a valid logic level at its input. 1 - invalid, 2 - valid logic levels. The settling time is usually not implemented by a delay line but by a clock cycle.

ABOUT THE AUTHOR

Wolfgang Matthes has developed peripheral subsystems for mainframe computers and researched special-purpose and universal computer architectures. He has also taught MCU Design, Computer Architecture, and Electronics at the University of Applied Sciences, Dortmund, Germany. He has filed over 50 patent applications and written 14 books. He is still busy writing books. You may reach him at: info@realcomputerarchitcture.com

D-type flip-flop connected to the asynchronous input signal and the synchronization clock (**Figure 36**).

The synchronization is a sampling process; the signal being sampled, for example, by the low-to-high edge of the synchronization clock. This clock edge hits, however, the asynchronous pulse train randomly. Therefore, various types of coincidence may occur, as illustrated in Figure 36b and **Figure 37**.

When the clock edge samples a signal level that does not change during the setup-hold-interval, nothing special will happen; the synchronized signal will take the logic level at the flip-flop's D input.

But what happens when the input signal of the flip-flop changes during the setup-hold interval surrounding the clock edge? Sometimes, nothing special will happen; the flip-flop will change its state or remain in the previous one.

However, there is a critical time interval within the setup-hold interval. Its width depends on the circuit technology and the structure of the flip-flop (we speak of picoseconds here; less than 1 to approximately 150ps are typical). If the input signal changes within this interval, the flip-flop may enter an intermediate state called the metastable state. In such a state, the flip-flop emits output signals that do not correspond to one of both logic levels (**Figure 38**).

Observe, the problem is not whether to catch a one or a zero. When edges coincide, the result of sampling is always by chance. To catch a pulse reliably, you must sample each of its levels (high and low) at least twice. TThis topic we will discuss in a forthcoming article. Instead, the problem is whether the flip-flop will assume one of the logic levels, may it be low, may it be high, or whether it will, so to speak, not know what to do and therefore output something in-between, as shown in Figure 38b, and 38c.

How probable is a metastable state? It could occur if the asynchronous input signal changes its level during a narrow critical time interval while the clock edge rises. In **Figure 39**, we give a first raw approximation, based on the probability of hitting an interval tW in an asynchronous pulse tAS wide. Here, we consider coincidences that may cause metastable states and hence failures. What matters above all is the average interval between two occurrences, the Mean Time Between Failures (MTBF).

The higher the signal and clock frequencies, the shorter the MTBF. The example in Figure 39 shows that even very humble frequencies (2MHz clock, 10kHz pulses are deemed slow today) will lead to an MTBF unacceptably short. The reason is that

19842860

we have not yet considered how metastable states can be remedied.

Metastable states are unavoidable. The principal remedy is to wait a particular time (settling time), hoping the metastable state has then vanished (**Figure 40**). Occasionally, it may happen that it does not vanish. This is deemed a failure. How often will it occur? The MTBF is not the average interval between occurrences of metastable states per se but between metastable states persisting after a particular settling time Δt has elapsed.

The delay line in Figure 40 is only meant as a symbol representing the settling time Δt. In practice, the delay between the sampling of the asynchronous and the processing of the synchronized signal is typically implemented by clocking. The most common solution is to sample with the downstream circuitry's clock.

Semiconductor manufacturers derive their MTBF data from experiments and simulations. The most significant formulas are depicted in **Figure 41**.

The formula (1) yields the MTBF as derived in Figure 39. But what about the width of the critical time window? It is a property of the flip-flop. Typically, it is described by an exponential function (2). Substituting it in (1) yields the ubiquitous MTBF formula (3). The parameters S and T depend on semiconductor technology and the flip-flop circuit. S has the dimension 1/ns or 1/s; T is in seconds (s). S

The Mean Time Between Failures (MTBF) as the reciprocal of the probability of a metastable state:

$$MTBF_{MS} = \frac{1}{f_C \cdot f_{AS}} \cdot \frac{1}{t_W} \quad (1)$$

The width of the critical time window as a property of the flip-flop, characterized by two parameters S, T:

$$t_W = \frac{T}{e^{\Delta t \cdot S}} \quad (2)$$

The basic formula to calculate the MTBF from the flip-flop's parameters S, T and the settling time Δt:

$$MTBF = \frac{e^{\Delta t \cdot S}}{f_C \cdot f_{AS} \cdot T} \quad (3)$$

The same, but with a time constant τ as the reciprocal of the parameter S:

$$MTBF = \frac{e^{\frac{\Delta t}{\tau}}}{f_C \cdot f_{AS} \cdot T} \quad (4)$$

The minimum settling time to ensure a stipulated MTBF:

$$\Delta t = \frac{\ln (MTBF \cdot f_C \cdot f_{AS} \cdot T)}{S} \quad (5)$$

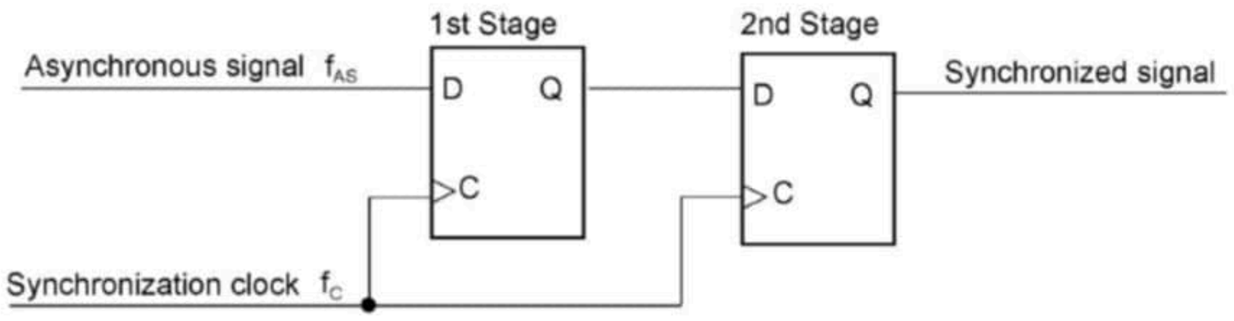
The MTBF of a two-stage synchronizer:

$$MTBF_2 = \frac{e^{2 \cdot \Delta t \cdot S}}{f_C^{\,2} \cdot f_{AS} \cdot T^2} \quad (6)$$

The MTBF of a three-stage synchronizer:

$$MTBF_2 = \frac{e^{3 \cdot \Delta t \cdot S}}{f_C^{\,3} \cdot f_{AS} \cdot T^3} \quad (7)$$

FIGURE 41
Formulas concerning synchronization and metastability.



The chance to fall into a metastable state repeats cyclically according to the clock frequency t_C. Therefore the probability of a metastable state (to be precise, the failure rate) is (as shown in Figure 39):

$$\frac{t_W}{t_{AS}} / t_C = \frac{t_W}{t_{AS} \cdot t_C} = f_{AS} \cdot f_C \cdot t_W \quad \text{with} \quad t_W = \frac{T}{e^{\Delta t \cdot S}}$$

A failure can only occur if both flip-flops fall into a metastable state. Each flip-flop has its corresponding probability.
The probability that two events occur together equals the product of the two independent probabilities.
The edges of the asynchronous signal and the output of the first stage must fall simultaneously in the critical window.

What has to coincide is:

- the edge of the asynchronous signal $\frac{1}{t_{AS}} = f_{AS}$
- the clock edge at the first stage $\frac{1}{t_C} = f_C$
- the clock edge at the second stage $\frac{1}{t_C} = f_C$
- the critical window of the first stage $t_{W1} = \frac{T}{e^{\Delta t \cdot S}}$
- the critical window of the second stage $t_{W2} = \frac{T}{e^{\Delta t \cdot S}}$

(Both flip-flops are assumed to be of the same type.)

All multiplied together:

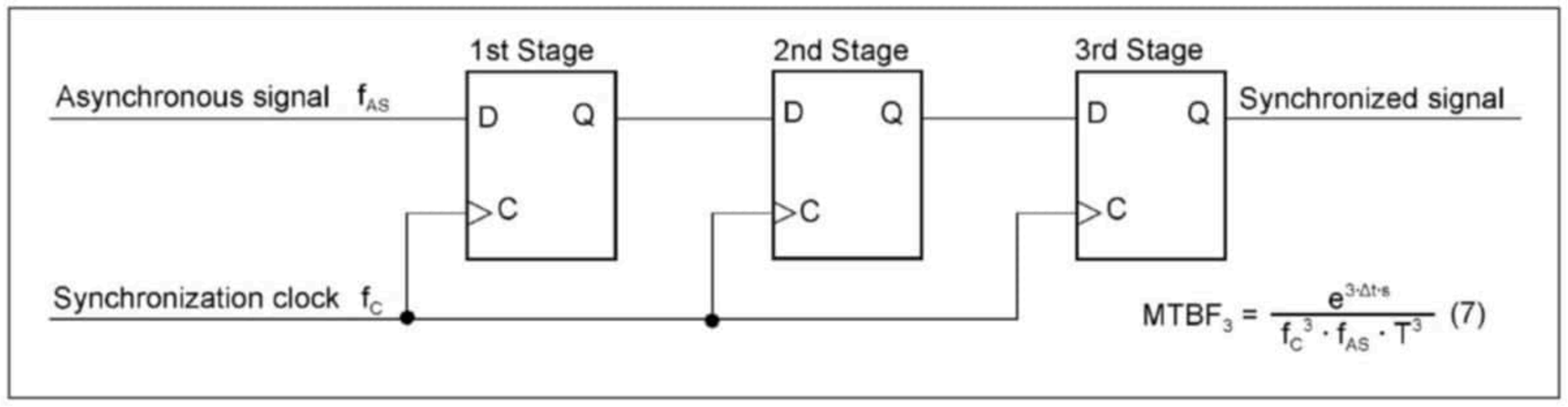
$$f_{AS} \cdot f_C \cdot f_C \cdot \frac{T}{e^{\Delta t \cdot S}} \cdot \frac{T}{e^{\Delta t \cdot S}}$$

The MTBF is the reciprocal of the failure rate:

$$MTBF_2 = \frac{e^{2 \cdot \Delta t \cdot S}}{f_C^{\,2} \cdot f_{AS} \cdot T^2} \quad (6)$$

FIGURE 42
Synchronization by two stages. The MTBF formula (6) is somewhat simplified here, assuming all flip-flops of the same type and one clock for all. In the literature, you will find more elaborate formulas. The principal structure is, however, the same.

FIGURE 43
Synchronization by three stages.



characterizes how fast the flip-flop will fall back into an acceptable state, T the width of the critical interval. S is determined by the gain-bandwith product of the feedback path of the flip-flop's master latch. Some manufacturers or authors specify a time constant τ which is the reciprocal of the parameter S. Then, the formula changes to (4).

The settling time Δt and the parameter S are parts of the exponent. Therefore, their influence at the MTBF is overwhelming. Prolonging the settling time slightly or selecting a flip-flop with a somewhat better S value will improve the MTBF considerably. IC designers strive for so-called metastable-resistant flip-flops showing large S or low τ values. Flip-flops expressly designed as metastable-resistant or even metastable-hardened, however, are larger and consume more current [4]. Thus, in today's FPGAs, special synchronizer flip-flops are missing. There are only two flip-flop types, one in the configurable logic blocks (CLB flip-flops) and one in the I/O blocks (IOB flip-flops); both being metastable-resistant to some degree.

SYNCHRONIZING TWO OR MORE

A failure is when metastable states affect the application circuitry. Metastable states internal of a synchronizer are not considered failures. Thus, a second synchronization flip-flop ensures that, when the first flip-flop assumes a metastable state, the application circuitry will not be affected. A failure will occur only if the second flip-flop also comes in a metastable state. By adding a second synchronization flip-flop, the MTBF can be increased significantly (**Figure 42**).

According to (6), the numerator of the MTBF formula increases vigorously (it is the numerator of the single-stage MTBF squared), while the denominator decreases because T is always < 1.

Therefore, at the expense of an additional clock cycle (and flip-flop, too), two-stage synchronization yields highly reliable circuits, even with flip-flops having comparatively low MTBF values. Some programmable circuits allow clocking the synchronization flip-flops with both edges (dualEDGE or DDR mode). Although the MTBF of the individual flip-flop is lower (due to the shorter clock cycles), the two-stage circuit's MTBF is significantly larger.

If the MTBF is still insufficient, more stages maybe the remedy. **Figure 43** shows a three-stage synchronizer. The MTBF formula (7) has been derived similarly to formula (6).

Synchronizer chains are shift-register-like flip-flop circuits as shown in Figure 42 and Figure 43. The chain ends with the flip-flop whose output is connected to another circuit. Within the chain, the connections between the flip-flop's outputs and inputs must not be tapped for other purposes.

SUMMARY

Today, digital design will be done mostly by instantiating readily designed subsystems as IP cores and by high-level behavioral descriptions. Such approaches, however, cannot cover all real-world problems.

To address real-world problems designers occasionally have to cope with synchronizing signals from outside, capturing pulses, suppressing glitches, and the like. Solving such problems requires familiarity with fundamental components. In Part 1 of this article series, we have concentrated on latches and flip-flops, sequential components that memorize single bits. In Part 2, we delved deeper into enabling or disabling flip-flops, timing parameters, and synchronization. In Part 3, I will look further into the problem of metastability and related subtleties of setting or resetting.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] to [6] as marked in the article can be found there.

RESOURCES

Arm | www.arm.com
Microchip | www.microchip.com
Xilinx | www.amd.com

embeddedworld

Exhibition&Conference

CONNECTING THE
EMBEDDED COMMUNITY

11.–13.3.2025
NÜRNBERG, GERMANY

Get your
free ticket now!
embedded-world.de/code
Use the voucher code **GGew25**

Medienpartner

elektroniknet.de
Markt&Technik
Elektronik
Elektronik automotive
Elektronik medical
connect professional

NÜRNBERG MESSE

Smart Home Lock Down

Matter Provides Security Blanket

TECH FEATURE

By
Tom Murphy

As more devices in the smart home connect to the Internet, they become increasingly vulnerable to outside attacks. Developers can now add the latest security measures to their Smart Home devices through Matter.

The old joke starts with the premise of scientists researching human behavior, investigating whether traits like optimism and pessimism are learned or inherited. The group begins to observe a field of younger students in laboratory settings. In one experiment, scientists place a child with noted signs of melancholy into a lab with a new set of the latest toys. In another lab, a child with outward indications of exuberance gets situated in a lab with only a pile of horse manure and a shovel. The scientists leave for 30 minutes and return to make observations.

In the lab with the new set of toys, researchers witnessed the child sitting alone in a corner with the toys untouched.

A scientist asked the student, "Did you know you are welcome to play with anything here?"

The child replied, "What's the use? They will all break within a half hour."

The lab with the horse excrement gave the scientists a different observation.

The test subject was having a ball using the shovel to sling the manure in all different directions.

A scientist asked the child, "You seem to enjoy yourself here. Can you tell us why?"

The child replied, "With all this poop, there's bound to be a pony around here somewhere."

The joke was related to give you a perspective on this month's *Circuit Cellar* editorial theme of Consumer Electronics with equal parts optimism and pessimism. With so many products and categories, how can journalists cover the subject without leaving something behind? So many important innovations today define new expectations and demand cycles for tomorrow. Besides, if I see one more set of "game-changing" earbuds, the trait of apathy might emerge.

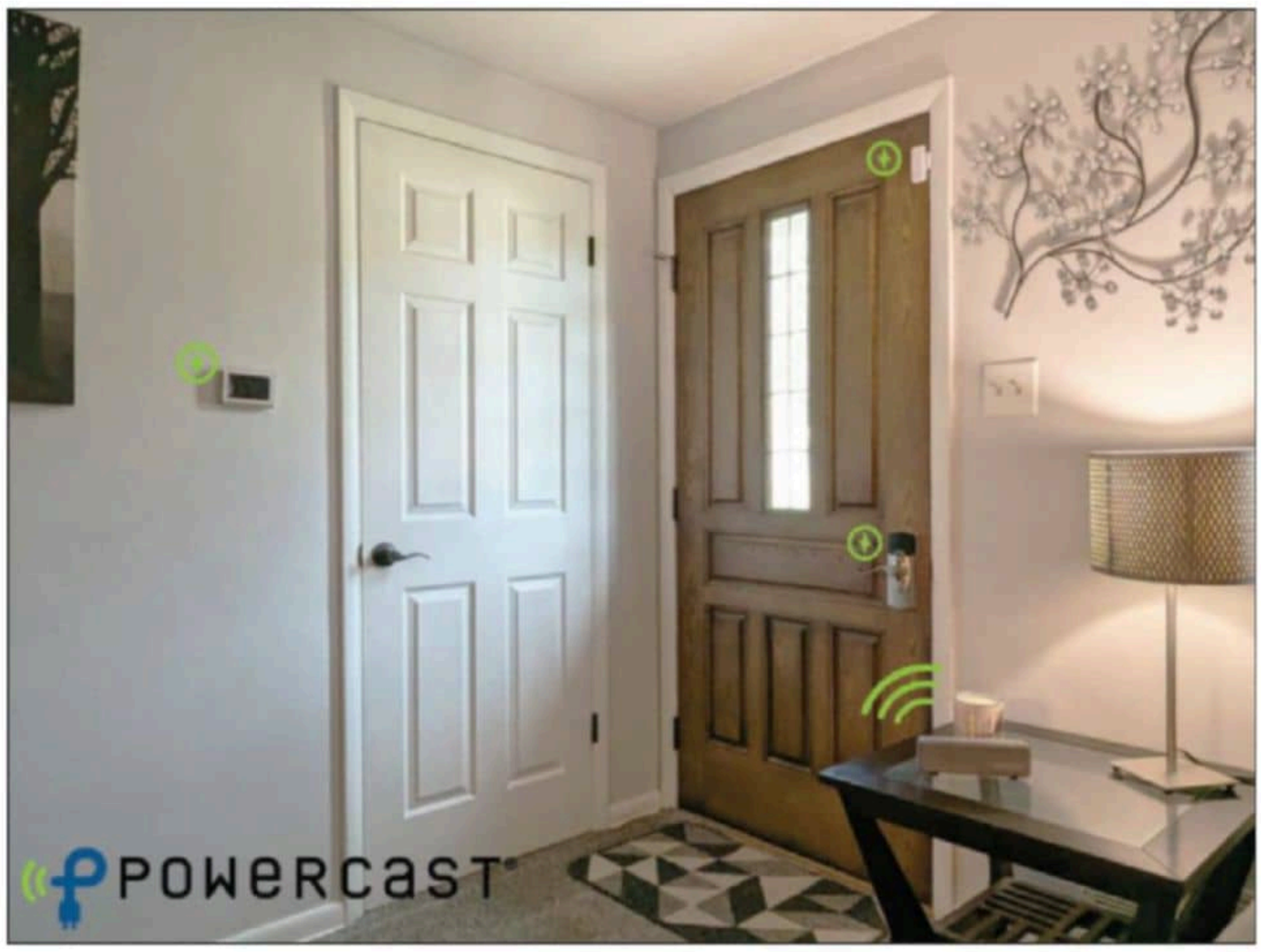


FIGURE 1
CES has recognized Powercast's efforts to reduce battery e-waste and power a greener future by bestowing the title of CES Innovation Awards 2025 Honoree in Sustainability. (Image source: Powercast)

What will be the hottest trend of the year in consumer electronics? Who knows? That could be like going to the beach and trying to collect a single, solitary drop of seawater.

I'm sure that there will be endless articles and media coverage devoted to identifying the top sellers in a range of categories and analyzing how they make life easier, more convenient, and entertaining.

But not here. *Circuit Cellar* defies convention in electronics coverage because there is possibly no way to look at the future and determine if something will be a hit. Sure, we could look at big brands like Nvidia and Samsung and try to triangulate the best estimates from an endless pool of experts and pundits. But blah, blah, blah.

Instead, we will look at a few aspects of the great wide world of CE and throw a dart or two. Wherever the darts land on the board will direct us to the "Thread" to follow and lead to "Matter" of fact conclusions.

FUTURE POWERCAST

Fortunately, one of the previous companies covered over the last 12 months won accolades at the 2025 International Consumer Electronics Show. [1]

Powercast, the wireless IoT charging company, was named an honoree for its home automation technology that powers up devices with something other than a battery. The company was chosen for its innovation in the sustainability/energy category for its Matter-compliant smart home energy sensor. Powercast was previously covered in *Circuit Cellar* issue 413 for its contributions to energy harvesting technology and then, prior to that, as part of Sensors Converge 2024 coverage.

The game changer here is the potential to reduce and eliminate batteries as an energy source for countless low-power IoT applications, including smart home automation applications like door locks, thermostats, remote controls, and anything related to items in **Figure 1**.

First, Powercast presents a way for the world to reduce its dependency on inexpensive lead-acid batteries. If a battery does not power your smart lock, then there will be one less cell in the parade of an estimated 5 million batteries that make it into landfills daily. That's one less battery leaking chemicals into the ground.

Second, it boosts the whole energy harvesting field. Energy harvesting technology has been proposed as a battery replacement application for a few decades. However, traditional ambient energy sources such as light, vibration, and heat have yet to sufficiently provide a consistent, reliable, and robust source for the myriad IoT devices running on batteries. Powercast offers the possibility of bridging the gap when energy harvesting is desirable but hardly practical. **Figure 2** shows one of the company's devices.

There are no hockey-stick-shaped growth curves for energy harvesting as a dynamic industry growth accelerator. Although a few research firms predict robust growth for this market segment, the transition from less than $1 billion to just over $1 billion could take until 2035.

WHAT'S THE MATTER?

Third, Powercast's honor is for "Matter" compliant wireless smart home automation sensors. Matter is a connectivity standard that ensures compatibility between Powercast's Powerharvester RF receivers and smart-home ecosystems such as Google Home, Amazon Alexa, and the Thread wireless network topology.

Powercast offers "install anywhere" convenience and eliminates the need for home wiring. It also purports to eliminate batteries and inconvenient, expensive maintenance for replacement. A Powerharverster receiver can be embedded into sensors to get power from nearby RF transmitters. One RF transmitter can power several sensors up to 25' away, so placing one in a room powers all RF-enabled sensors for windows, doors, temperature, light, humidity, water, and motion.

Matter and "Thread" make Powercast's entry into this year's CES Innovation Awards interesting. Matter, as seen in

FIGURE 2
Manufacturers embed a Powerharvester wireless RF receiver chip and an antenna into their sensors to be powered wirelessly over the air from nearby RF transmitters. (Image source: Powercast)

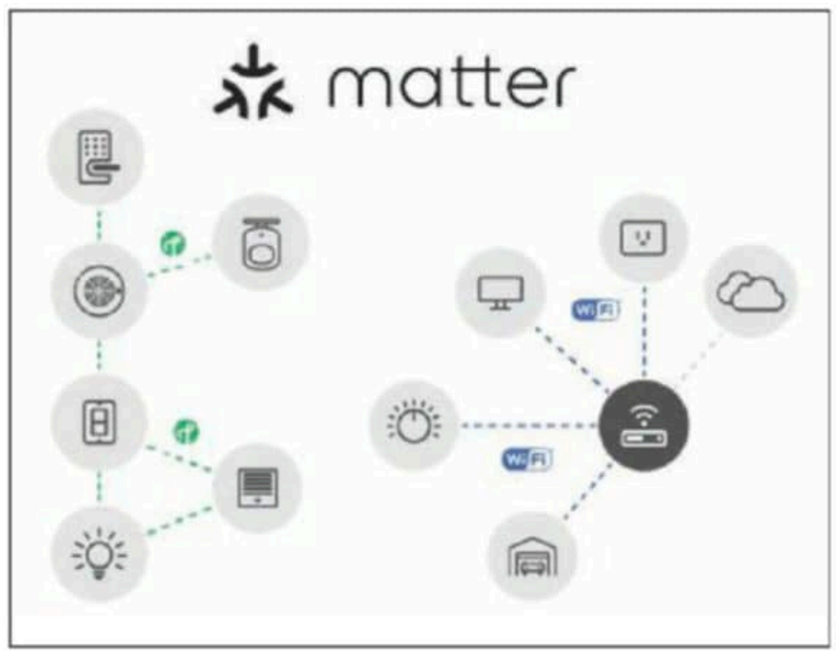


FIGURE 3
The CSA-IoT has defined Matter as a way for smart home devices to interoperate securely. (Image source: Silicon Labs)

Figure 3 is the wireless protocol developed by the Connectivity Standard Alliance (CSA-IoT) [2] to promote interoperability. The CSA touts Matter as one protocol for connecting compatible devices and systems.

The goal is to make smart home devices secure, reliable, and seamless. By the way, Thread is a reliable, secure, and energy-saving network protocol that delivers fast response times and extended coverage to elevate smart home experiences, according to the Thread Group's home pages [3]. **Figure 4** shows more about this network topology.

The efforts are designed as unifying standards, promising reliable, secure connectivity. It is a seal of approval that devices will work seamlessly together. Matter creates more connections between more objects, simplifies development for manufacturers, and increases consumer compatibility.

COME TOGETHER

For developers, Matter is a unifying, IP-based connectivity protocol built on proven technologies, helping them to connect to and build reliable, secure IoT ecosystems and keeping them focused on developing innovative products and accelerating paths to market, according to the CSA-IoT [3]. As a unified, open-source application-layer connectivity standard built to enable developers and device manufacturers to connect and build reliable and secure ecosystems and increase compatibility among connected home devices.

It is built using Internet Protocol (IP) and is compatible with Thread and Wi-Fi network transports. Matter was developed by a working group within the Connectivity Standards Alliance. This Working Group develops and promotes the adoption of royalty-free connectivity standards to increase compatibility with security as a fundamental design tenet. The vision that led major industry players to build Matter is that smart connectivity should be simple, reliable, and interoperable.

Matter simplifies development for manufacturers and increases compatibility for consumers.

By building upon IP, Matter enables communication across smart home devices, mobile apps, and cloud services and defines a specific set of IP-based networking technologies for device certification. The Matter specification details everything necessary to implement a

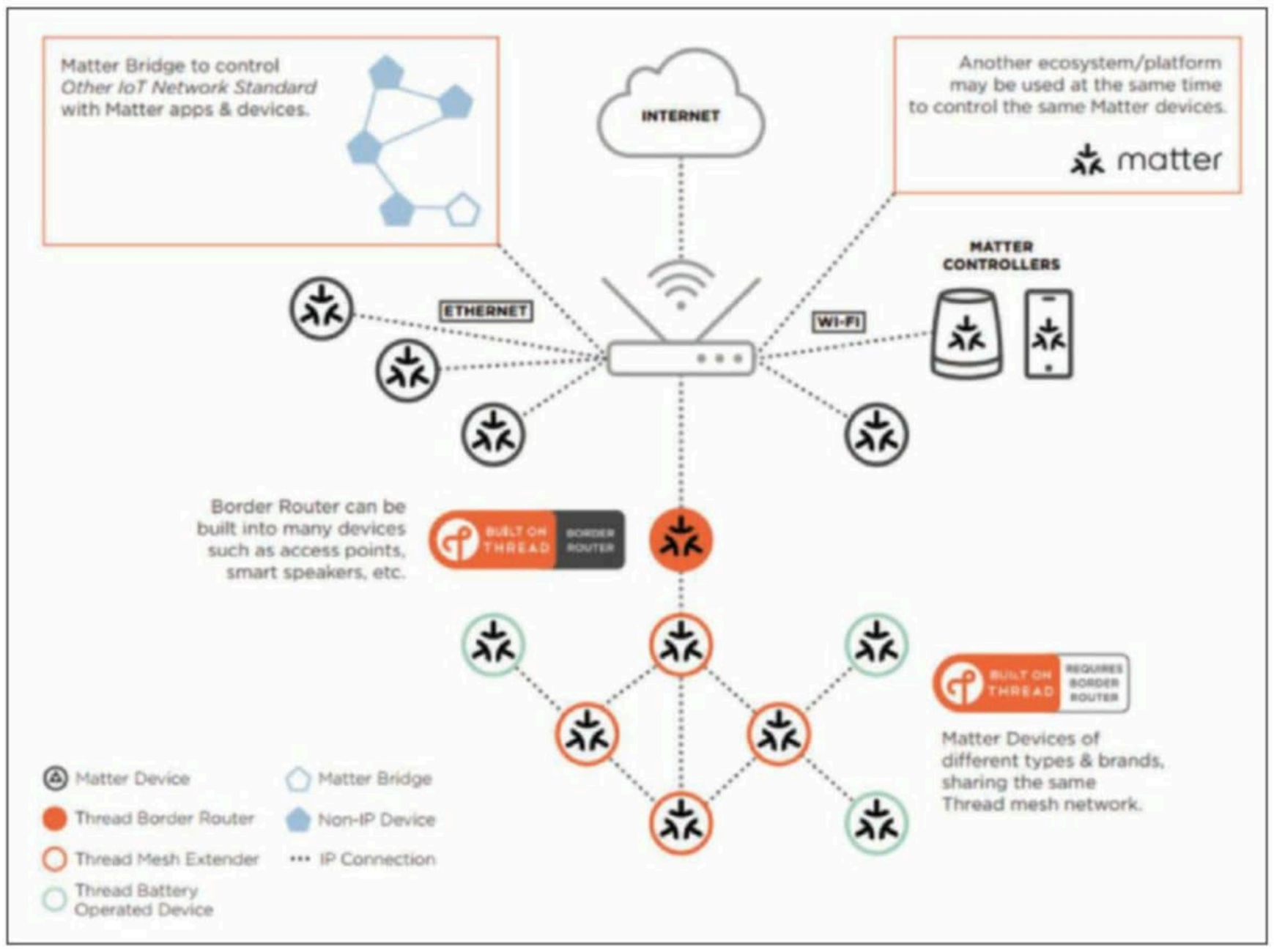


FIGURE 4
The Thread network transport layer is depicted as it relates to Matter. (Image source: Thread Group)

Matter application and transport layer stack. It is intended to be used by implementers as a complete specification.

Matter is developed with the following goals [4] and principles in mind according to a GitHub repository:

Unification: Matter is built with and on top of market-tested, existing technologies.

Interoperability: The specification permits communication between any Matter-certified device, subject to users' permission.

Security: Leverages the latest security practices and protocols.

User Control: The end user controls authorization for interaction with devices.

Federated: No single entity serves as a throttle or a single point of failure for root of trust.

Robust: The set of protocols specifies a complete lifecycle of a device starting with the seamless out-of-box experience, through operational protocols, to device and system management specifications required for proper function in the presence of change.

Low Overhead: The protocols are practically implementable on low compute-resource devices, such as MCUs.

Pervasive: The protocols are broadly deployable and accessible by leveraging IP and being implementable on low-capability devices.

Ecosystem-Flexibility: The protocol is flexible enough to accommodate deployment in ecosystems with differing policies.

Ease-of-Use: The protocol provides a smooth, cohesive, integrated provisioning and out-of-box experience.

Open: The project's design and technical processes are open and transparent to the general public, including non-members, wherever possible.

Matter supplies a universal IPv6-based communication protocol for smart home devices, as shown in **Figure 5**. The protocol defines the application layer deployed on devices and the different link layers to help maintain interoperability.

The architecture is divided into layers, as shown in **Figure 6**, to help separate the different responsibilities and introduce a level of encapsulation among the various pieces of the protocol stack.

Application: The high-order business logic of a device. For example, an application focused on lighting might contain logic to handle turning on/off the bulb and its color characteristics.

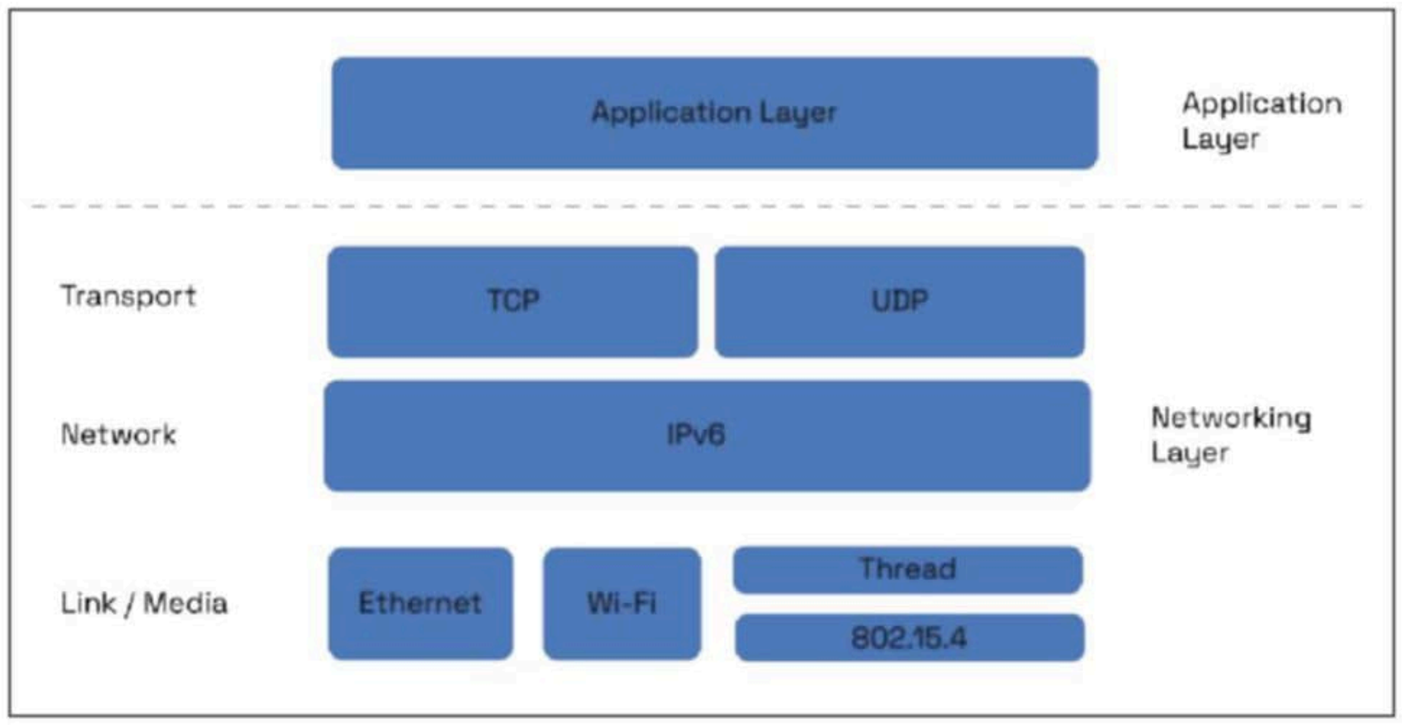


FIGURE 5
The following diagram illustrates the normal operational mode of the Matter stack. (Image source: Matter Git Hub.)

Data Model: The data layer corresponds to the data and verb elements that help support the functionality of the application. The application operates on these data structures when there is an intent to interact with the device.

Interaction Model: This layer defines a set of interactions that can be performed between a client and server device. For example, reading or writing attributes on a server device would correspond to application behavior on the device. These interactions operate on the elements defined at the data model layer.

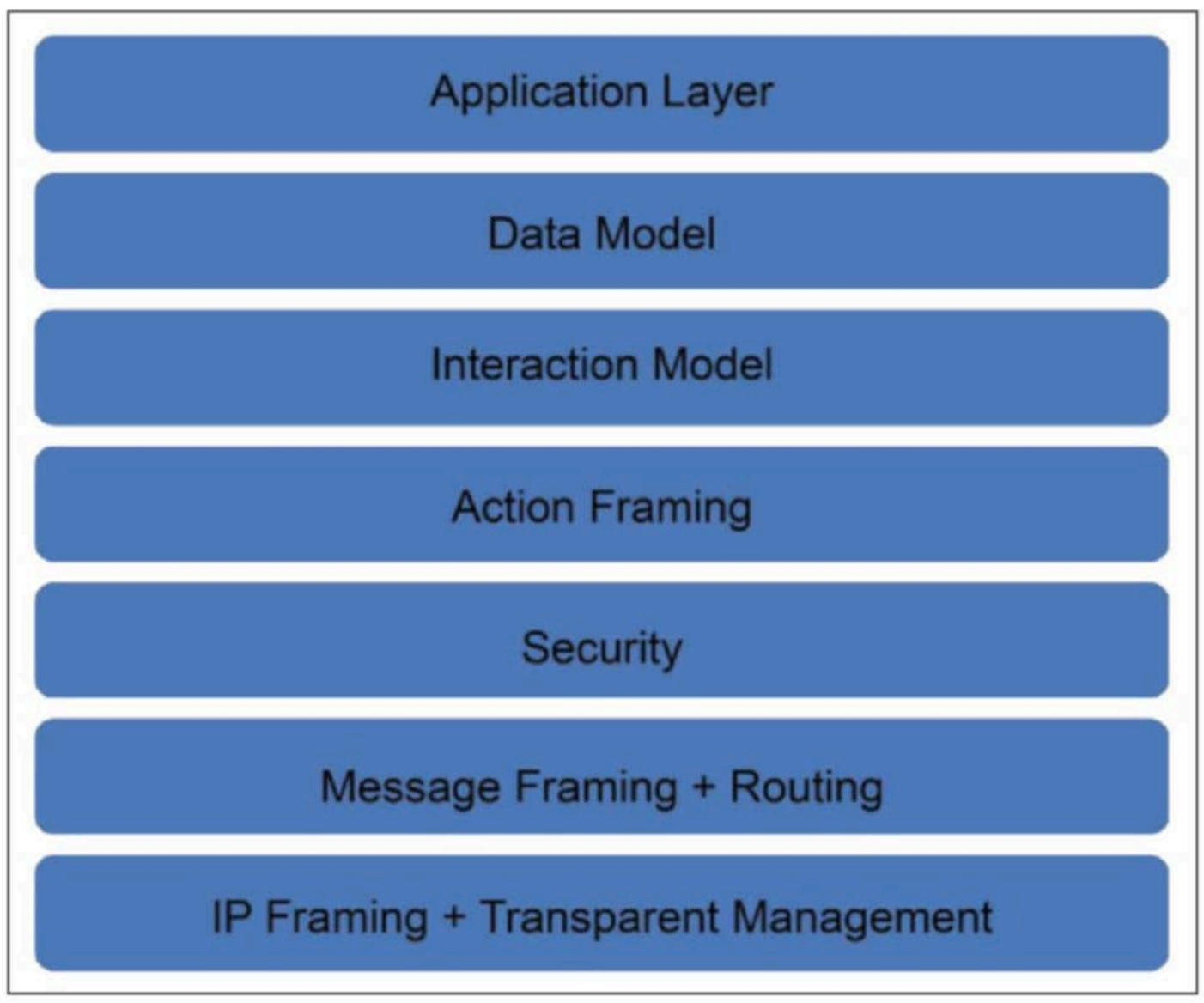


FIGURE 6
The vast majority of Matter interactions flow through the stack shown here. (Image source: Matter)

FIGURE 7
What if Waffle Bot, hero of the motion picture, "A Very Harold and Kumar Christmas" were to come to life?

Action Framing: Once an action is constructed, it is serialized into a prescribed packed binary format to encode for network transmission.

Security: An encoded action frame is then sent down to the security layer to encrypt and sign the payload, ensuring that data is secured and authenticated by both sender and receiver of a packet.

Message Framing and Routing: With an interaction encrypted and signed, the Message Layer constructs the payload format with required and optional header fields, which specify the message's properties and some routing information.

IP Framing & Transport Management: After the final payload has been constructed, it is sent to the underlying transport protocol for IP data management.

Additional materials from the author are available at: www.circuitcellar.com/article-materials

References [1] to [6] as marked in the article can be found there.

Resources

Powercast | www.powercastco.com

Connectivity Standards Alliance | www.csa-iot.org

Infineon | www.infineon.com

Turtlebot | www.turtlebot.com

SECURING IOT AT HOME

The research on Matter led to a serendipitous discussion with Steve Hanna, Distinguished Engineer, Infineon Americas on security for IoT devices.

Hanna emphasizes the security measures that device makers embed in their devices because attacks are rising. Hanna and Infineon emphasize a Zero Trust Architecture scheme that offers devices the level of security in this brave new world of emboldened attackers.

"The risk doesn't come from within the home but from outside. Many smart devices are connected to Wi-Fi, and that is where the risk comes into the home," Hanna said in a recent conversation with *Circuit Cellar*.

"If you make a smart device and connect it to a network, even Bluetooth, and certainly Wi-Fi, you are adding a big element of risk," Hanna said. "If you think the bad guys are outside your home, that is the natural thing to think. The fact is they can get inside your home and infect a Wi-Fi router if the patches aren't up-to-date. It could be from downloading games, and along with the games comes some malware."

Once inside a home, the scenario could resemble a horror movie.

"Now, the Wi-Fi network has a malicious code running on it," Hanna continued. "We saw this 10 years ago with the Marai Bot Net, where millions of Wi-Fi routers are enslaved into a bot net, they're infected. That malicious code links back to a controller, and the bad actor can control all these little zombies around the world and use them for all the purposes they want.

"As designer of IoT systems, we have to assume that the bad actors are on the Wi-Fi network. In a home with 10 to 20 or 100 different IoT devices. Some of those will always be infected, and they will never get down to zero."

Of course, a company like Infineon has the resources for a giant multinational corporation to address malicious actors with the various security schemes embedded in hardware. A person making their own IoT device might not have the time or resources to ensure all the security measures are in place before linking a project to the net.

Fortunately, that's where Matter comes back into relevancy.

"Take a look at Build With Matter.com [4]," Hanna said. "From there, learn how to use the Matter SDK. It runs on the Raspberry Pi and other common development platforms supported by Infineon and many other chip makers. You can create all your IoT devices yourself, you can take advantage of what's already made in Matter. We build into Matter the best-of-breed security that we could all come up with.

"Matter 1.0 had 10 added security features that hadn't been in the smart home. Every year we are going to be adding new features. You never have enough security. The bad guys are always figuring out ways to break in.

"We don't want to add too much security to make it too expensive. We want to add enough layers to deter the bad guys. As a weekend warrior setting cool smart home devices, Matter is the easy way to add security to your device."

WHAT WOULD A DEVELOPER DO?

The beginning of the new year promises a new challenge. The editorial staff consists largely of liberal arts majors and publishing types who have limited exposure to the engineering tasks and skills needed to design an embedded system. But that's all going to change, as one editor has promised to get some parts and assemble something themselves as a learning project.

This bright idea came during a search for a gift for a relative who is always difficult to shop for, and rather than issue the usual gift card, a bright idea came to mind. "Oh, I know, let's go on eBay and find him 'Waffle Bot,' " this editor told himself. (**Figure 7**)

Waffle Bot is the scene-stealing electronic sidekick in the 2011 movie *A Very Harold and Kumar Christmas* starring Kal Penn and John Cho. [5]

In it, one of the main protagonists and Waffle Bot bond over their mutual love of the baked batter breakfast confection. Without spoiling the whole plot, Waffle Bot does come to the rescue during unforeseen circumstances with often hilarious results. Please excuse the digression here.

A relative also mentioned Waffle Bot during a conversation, so an inexpensive movie prop available for online purchase would make a great gift.

Nope. Not an option. What would a developer do in such a circumstance? Build his own.

The decision was made to build one from parts.

I chose Legos to form the basis of the actual Waffle Bot, including its body, arms, and head, and an actual robot kit for the device's innerworkings.

I chose the TurtleBot3 Waffle Pi [6] because its title includes " waffle." See **Figure 8**.

Additionally, it has the following features:

- ROS Compatibility is a ROS-programmable platform that offers a complete autonomous navigation system.
- A Raspberry Pi 3 Centric System that incorporates superior computing power and connectivity for an enhanced robotics experience.
- Raspberry Pi Camera Module enables real-time viewing and recording, which is ideal for surveillance or exploration applications.
- Dynamixel servos come equipped with TTL-connected servos for greater precision and movement reliability.

FIGURE 8
The Robotis Turtlebot3 Waffle Pi comes with LiDAR. (Image source: Robotis)

With dimensions of 281mm x 306mm x 141mm and a weight of 1.8kg, the Waffle Pi is designed to be both robust and easy to handle.

Movement Capabilities: Maximum translational speed of 0.26m/s and rotational speed of 1.82rad/s for exceptional maneuverability.

Approximately two hours of continuous use with a 2.5-hour recharge, ensuring an extended experience.

Integrated sensors include a 360° LiDAR, gyroscope, accelerometer, and magnetometer for complete environmental perception.

The sales literature reveals the following facts under the hood: "The TurtleBot3 Waffle Pi represents a significant advance in programmable robotics. Its integration of the Raspberry Pi 3 offers a powerful and versatile platform suitable for a wide range of educational and research applications. Whether you're an enthusiastic beginner or an experienced professional, the Waffle Pi is an invaluable tool for exploring the infinite potential of robotics.

This is one liberal arts major who is going to take the Lego Waffle Bot ensemble and a hot glue gun to firmly affix it to the Waffle Pi—at least, that's the plan right now. Please check in future issues for hilarious, if not tragic, results.

Datasheet:
Microamps Per Megahertz

Ultra-Low Power MCUs Minimize Current Consumption

By
Tom Murphy,
Editor-in-Chief

DATASHEET

How do chip makers differentiate if many ultra-low power MCUs on the market feature the same processor core? The peripherals and different power states offer various ways to manage current consumption down to microamps per megahertz.

What criteria do embedded developers and chip makers use to determine what makes a microcontroller qualify as an ultra-low-power device? For some MCU architectures, low power consumption is intrinsic by nature due to relatively small size, low-frequency cores, and minimal memory resources. The Arm Cortex-M4F core is one that comes to mind, and in this month's Datasheet, the architecture is well represented among nine different selections.

The difference between a low-power and an ultra-low-power microcontroller SoC seems to lie within the peripherals around the cores and the schemes that they use to put the device in a deep sleep cycle when it is inactive. Scanning the differences between nine different ultra-low-power offerings offers insight into the key metrics that manufacturers want to instill with developers: microamps per megahertz (µA/MHz).

One of the datasheets profiled offers this metric: 0.20µA/MHz in VDD dynamic current sleep mode. The number itself seems impressive from a power-sipping perspective. But when you read the words after it, "dynamic current sleep mode," questions arise about what exactly that means.

Do we have to program the device to reach that level, or does an embedded scheme figure that out through some smart current sensing apparatus? Does that apparatus carry some current consumption penalty? Of course, it all depends. Where will the device be used? Does it process intermittent sensor activity or deliver peak current from power up to power off?

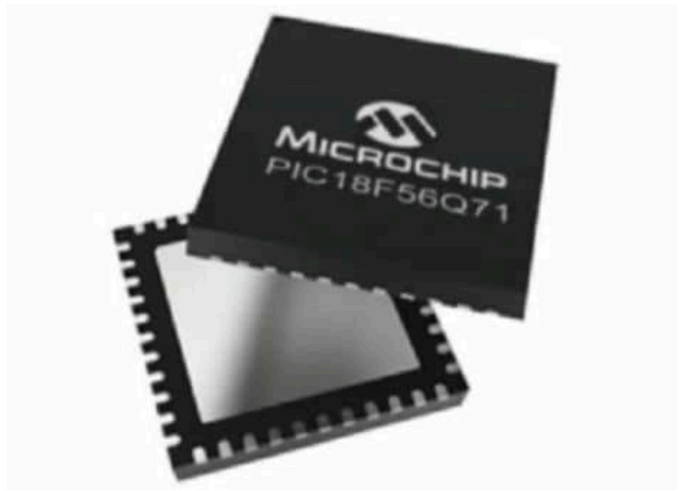
When the processor core is the same across manufacturers, the competitive advantage comes from the peripherals surrounding the core and how it manages different states of operation when the device is at rest, how soon it needs to wake up, and how it delivers full power when needed.

If a processor core can be put to sleep, how will it know when to wake up to detect an event that it needs to record, process, and send a packet to the cloud? Various schemes are proposed, including one where the processor core stops spinning in a semi-inactive state. Still, the analog portion remains in a semi-awake state in case such an event triggers a sensor reading or alarm.

In these schemes, a processor can shift from deep sleep mode to fully active mode in milliseconds. It can also maintain memory power to ensure no data loss between cycles. In some datasheets presented below, cores can shuffle through as many as five power states before settling back down to "Deep Sleep."

Still, other schemes to achieve ultra-low power rely on integrating components other than the MCU core on the premise that it is more efficient to perform a task on-chip than off the SoC die.

NEXT MONTH'S TOPIC: Modular System Boards Send related product announcements to editor@circuitcellar.com



“World’s Smallest” ULP MCU

Made for wearables, hearables and the like, Analog Devices’ MAX32660 boasts one ultra-low power number: 0.20µA/MHz in VDD dynamic current sleep mode. It gives users the maximum amount of operating time between battery charges, and the Arm Cortex-M4 32-bit microcontroller, 96KB SRAM and 256KB Flash memory are packed into a 1.6mm x 1.6mm, 16-bump WLP.

- Internal oscillator to 96MHz
- 256KB Flash memory
- 96KB SRAM in low power backup mode
- 16KB instruction cache
- Memory protection unit (MPU)
- Low 1.1V VCORE supply voltage
- 3.6V GPIO operating range
- Internal LDO
- -40°C to +105°C temperature range
- 85µA/MHz active executing from flash
- 2µA full memory retention power
- Backup mode at VDD = 1.8V
- 450nA ultra-low power RTC
- Internal 80kHz ring oscillator
- Optimal peripheral mix
- Platform scalability
- Up to 14 GPIO Pins
- Up to two SPI
- I2S
- Two UARTs
- Two high-speed I2C at 3.4Mbps
- Four-channel DMA controller
- Three 32-bit timers
- Watchdog timer
- CMOS-level 32.768kHz RTC output
- 4mm x 4mm, 20-pin TQFN-EP
- 3mm x 3mm, 24-pin TQFN-EP
- Sports watches
- IoT

Analog Devices
www.analog.com

Infineon PSoC 6

The Infineon PSoC 6 MCU is a high-performance, ultra-low-power and secure platform, purpose-built for IoT applications and based on the ARM Cortex-M4F. The CY8C61x4 product line offers digital programmable logic, high-performance analog-to-digital conversion and a rich set of analog peripherals that operate in device deep sleep mode for low-power analog signal processing. Ideal for battery-powered applications, low-power sensor processing, motor control, and more.

- 150MHz Arm Cortex-M4F
- Arm Cortex-M0+ reserved for system functions
- Ultra-low power 40nm platform
- Ultra-low power (0.9V) and low power (1.1V) core voltage operation
- 6 power management modes
- Active CPU current slope
- Cortex-M4: 40µA/MHz
  Cortex-M0+: 20µA/MHz
- Active CPU current slope with 0.9V core operation
- Cortex-M4: 22µA/MHz
  Cortex-M0+: 15µA/MHz
- Deep Sleep mode current of 7µA
- 64KB SRAM retention
- On-chip DC-DC Buck converter
- <1µA quiescent current
- 256KB flash
- 128KB SRAM
- 1.7V to 3.6V operation
- On-chip DC-DC buck converter
- 2 12-bit 2-MSPS SAR ADCs
- Packages: 64-TQFP (54 IOs),
- 68-QFN (52 IOs)
- 80-TQFP (62 IOs)

Infineon
www.infineon.com

Microchip PIC18-Q71

Microchip offers configurable, analog-focused MCUs for sensor interfacing designs with the PIC18-Q71 product family. Ultra-low-power operation achieved through Core Independent Peripherals (CIPs) and a high level of analog integration to simplify sensor interfacing and analog measurements, optimize system performance and reduce BOM cost. Made for LED lighting, predictive maintenance, medical, home automation, industrial process control, automotive and IoT.

- Operating Speed DC: 64MHz clock input
- Operating Range: – 1.8V to 5.5V
- -40°C to 85°C temperature range
- Low-Current Power-on Reset (POR)
- Configurable Power-up Timer (PWRT)
- Brown-out Reset (BOR)
- Up to 64KB of Program Flash Memory
- Up to 4KB of data SRAM Memory
- 256 Bytes data EEPROM
- Power-saving functionality
- Doze: Different cycle rates for CPU and Peripherals
- Idle: CPU halted, peripherals operate
- Sleep: Lowest power consumption
- Peripheral Module Disable (PMD)
- Selectively disable hardware
- Minimize active power consumption
- Analog peripheral manager
- Optimize power consumption
- Low-power mode features
- Sleep: < 1µA typical @ 3V
- 48µA @ 32kHz, 3V, typical operating current

Microchip
www.microchip.com

DATASHEET

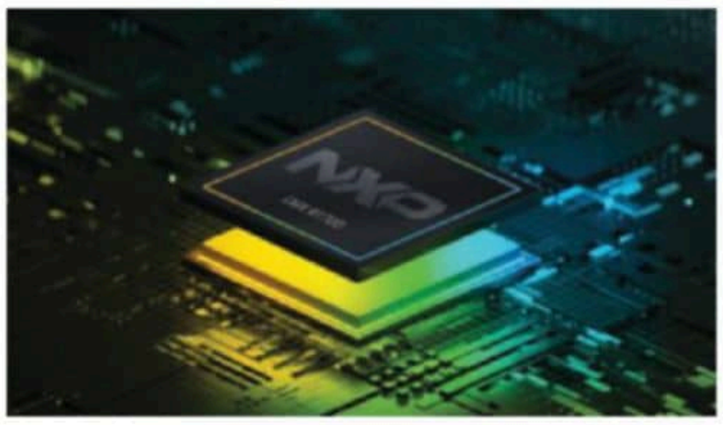
DATASHEET URLS:

Analog Devices: https://www.analog.com/media/en/technical-documentation/data-sheets/max32660.pdf

Infineon: www.infineon.com/dgdl/Infineon-PSoC_6_MCU_CY8C61X4-DataSheet-v06_00-EN.pdf?fileId=8ac78c8c7ddc01d7017ddd0276175906

Microchip: https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/PIC18F26-46-56Q71-Microcontroller-Data-Sheet-DS40002329.pdf

Datasheet:
Microamps Per Megahertz



AI-Colored Glasses

NXP is in pre-production with the i.MX RT700 that features up to five computing cores designed to power AI-enabled edge devices such as wearables, consumer medical, smart home, and HMI devices. Ultra-low power features come via "Sense Subsystem" that includes a second Arm Cortex-M33 and Cadence Tensilica HiFi 1 DSP, replacing an external sensor hub and reducing system design complexity, footprint, and BOM costs.

- Arm Cortex-M33 up to 325MHz
- HiFi 4 DSP up to 325MHz
- eIQ Neutron NPU up to 325MHz
- Sense Compute subsystem
- Arm Cortex-M33 up to 250MHz
- HiFi 1 DSP up to 250MHz
- 7.5MB on-chip SRAM
- Three xSPI interfaces
- Off-chip memory expansion
- Up to 16b-wide external memories
- Up to 250MHz DDR
- Two SD/eMMC memory card interfaces
- eMMC 5.0 with HS400/DDR operation
- USB high-speed host/device controller
- Eight-channel microphone interface
- Serial peripherals (UART/I²C/I3C/SPI/HSPI/SAI)
- 2.5D GPU with vector graphics acceleration
- SIMD/DSP instructions
- Up to 720p@60FPS
- CSI 8/10/16-bit parallel (via FlexIO)
- EdgeLock Secure Enclave
- Secure boot and debug
- Crypto-TRNG, DICE, UID, PUF, OTP
- Crypto accelerators for PKC, AES, SHA
- Tamper and intrusion detection

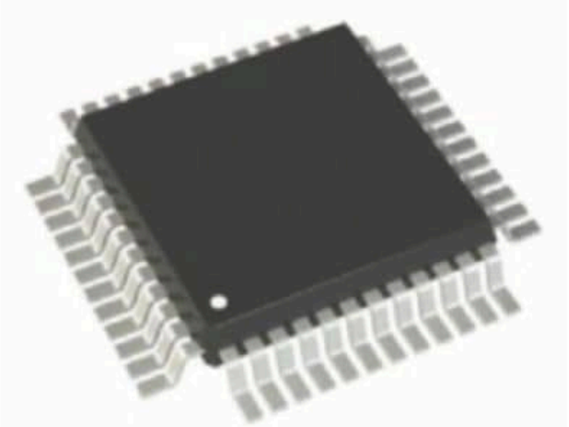
NXP
www.nxp.com

Qorvo Motor Control

The Qorvo PAC55710 is a 70V BLDC motor controller and driver that offers Vds Sensing, Cycle By Cycle (CBC), Enhanced Sample And Hold (S&H), Windowed Watchdog Timer for controlling and powering smart energy appliances, devices, and equipment. The SoC integrates a 150MHz Arm Cortex-M4F 32-bit microcontroller core with Qorvo's proprietary and patent-pending Multi-Mode Power Manager, configurable analog front end, and application specific power drivers.

- 150MHz Arm Cortex M4F
- 128kB Flash
- 32kB SRAM
- 70V Buck/SEPIC DC/DC Controller
- 5V/200mA system supply
- 3 High-side 70V 1.2A (source)/1.8A (sink) gate drivers
- 3 Low-side 20V 1.2A (source)/1.8A (sink) gate drivers
- Vds Sensing
- Advanced Cycle-by-Cycle Protection
- 3 Differential PGA (x1 to x48)
- 4 Single PGA (x1 to x48)
- Programmable over-current shut-down
- High-speed 12-bit 2.5MSPS ADC
- with dual auto-sampling sequencers
- 3.3V I/Os,
- Simultaneous sample and hold for three-phase
- 5.2V to 70V input voltage range
- UART/SPI, I²C, and CAN serial interfaces
- 25x MCU GPIOs
- 48-pin TQFN package, 6mm x 6mm

Qorvo
www.qorvo.com



Ultra-Low Renesas

The Renesas RA0E1 offers ultra-low power consumption, up to 32MHz of CPU performance with an Arm Cortex-M23 core, 64KB flash, 12KB SRAM and a wide supply voltage range from 1.6V to 5.5V. The RA0 devices consume only 84.3µA/MHz of current in active mode and only 0.82mA in sleep mode. Designed for low-power consumer electronics, small appliance system control, industrial system control, and building automation.

- 32MHz Arm Cortex-M23 core
- 64KB Flash
- 12KB SRAM
- 1KB data flash memory
- 16-pin and 32-pin package options
- 1.6V to 5.5V voltage range
- TAU 16-bit general PWM timer
- TML32 32-bit interval timers
- RTC
- 12-bit A/D converter
- Temperature sensor
- SAU (UART, Simple SPI, Simple I2C)
- UART
- Random number generator
- Flash area protection
- Illegal memory access detection
- ±1% high accuracy oscillator
- Flexible power architecture
- System power management
- Low power modes
- Low Voltage Detection (LVD)
- Fast wake-up
- -40°C to +105°C temperature range
- 16-pin HWQFN (3 mm × 3 mm, 0.5 mm pitch)
- 32-pin LQFP (7 mm × 7 mm, 0.8 mm pitch)

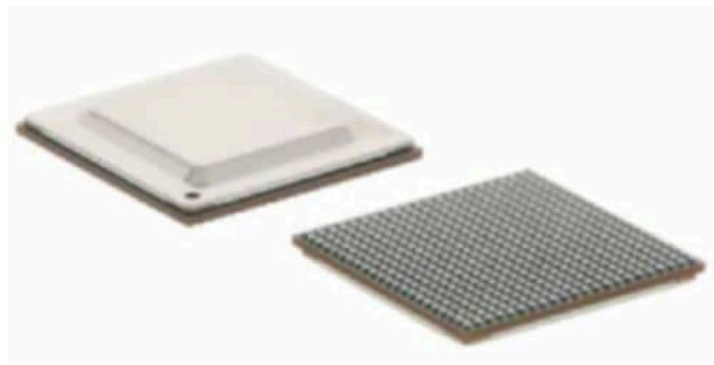
Renesas
www.renesas.com

DATASHEET URLS:

NXP: https://www.nxp.com/products/i.MX-RT700?tid=vanimxrt700

Qorvo: https://www.qorvo.com/products/d/da009499

Renesas: https://www.renesas.com/en/document/dst/ra0e1-group-datasheet



Power So Low, It's Silly

The Silicon Labs series of EFM32PG23 Gecko MCUs is intended for energy-friendly embedded applications. The ultra-low power package has an 80MHz Cortex-M33 with analog and communication peripherals and a low 21µA/MHz in Active Mode for in metering, industrial automation, test and measurement, appliance, and portable medical device applications.

- 32-bit 80MHz ARM Cortex-M33
- DSP instruction and FPU
- Up to 512kB flash program memory
- Up to 64kB RAM data memory
- Low Energy Consumption
- 21µA/MHz in Active Mode at 80MHz
- 1.33µA EM2 DeepSleep current
- 64KB RAM retention and RTC
- 1.03µA EM2 DeepSleep current
- 16KB RAM retention and RTC
- 1.71V to 3.8V operating range
- -40°C to 125°C
- QFN40 5mm × 5 mm × 0.85 mm
- QFN48 6mm × 6mm × 0.85mm
- High speed mode up to 2Msps
- High accuracy mode up to 16-bits ENOB at 3.8ksps
- 2 Analog comparators (ACMP)
- Two-Channel Digital-to-Analog Converter (VDAC)
- Low-Energy Sensor Interface (LESENSE)
- Up to 34 General Purpose I/O pins
- 3 EUSART
- UART/SPI/Smart Card
- 2 I2C interface with SMBus support
- HW cryptographic acceleration
- TRNG
- 34 GPIO pins

Silicon Labs
www.silabs.com

Alarmingly Low Power

The STM32L0 series is STMicroelectronics' entry-level range of 32-bit MCUs designed for ultra-low power consumption levels as low as 49µA/MHz with external DC/DC. With dynamic voltage scaling, the STM32L051x6 with an Arm-based Cortex-M0+, with 64KB Flash, 8KB SRAM, 2KB EEPROM, and an ADC is aimed at energy harvesting, smoke detectors, gas and water meters and consumer devices with advanced battery management and fast charging.

- Ultra-low-power platform
- 1.65V to 3.6V power supply
- -40°C to 125 °C temperature range
- 0.27µA Standby mode (2 wakeup pins)
- 0.4µA Stop mode (16 wakeup lines)
- 0.8µA Stop mode + RTC + 8-Kbyte RAM
- retention
- 88µA/MHz in Run mode
- 3.5µs wakeup time (from RAM)
- 5µs wakeup time (from Flash)
- 340nA ULP mode/full RAM/lp timer
- Arm 32-bit Cortex-M0+ with MPU
- 32kHz up to 32MHz maximum
- 0.95 DMIPS/MHz
- 64KB Flash memory
- 8KB RAM
- 2KB of data EEPROM
- 51 fast I/Os
- 45 I/Os 5V tolerant
- Ultra-safe, low-power BOR
- Ultra-low-power POR/PDR
- 1MHz to 25MHz crystal oscillator
- 32kHz oscillator for RTC
- 12-bit ADC 1.14Msps up to 16 channels (down to 1.65V)

STMicroelectronics
www.st.com

64-Bit Display Play

Power-optimized and efficient for display and graphics applications, the Texas Instruments AM62x brings up to four 64-bit Arm Cortex-A53s. The jump up from a 32-bit architecture doesn't necessarily mean exorbitant power consumption—rather it enables dynamic control of system resources with finer granularity. The AM62x is certified for automotive systems where power demands are many and supply is finite.

- Quad-core Cortex-A53 cluster
- 512KB L2 shared cache
- With SECDED ECC
- 32KB L1 DCache per A53 Core
- Dedicated device power manager
- Partial I/O support for CAN/GPIO/UART wakeup
- DeepSleep
- MCU Only
- Standby
- Dynamic frequency scaling for Cortex-A53
- Dual display support
- 1920x1080 @ 60fps for each display
- 1x 2048x1080 + 1x 1280x720
- OLDI (4 lanes LVDS - 2x)
- 3D Graphics Processing Unit
- Fillrate greater than 500Mpixels/sec
- >500MTexels/sec, >8 GFLOPs
- 60fps
- One Camera Serial interface (CSI-Rx)
- MIPI CSI-2 v1.3 Compliant + MIPI D-PHY 1.2
- Arm Cortex-M4F MCU at up to 400MHz
- 16nm technology
- 13mm x 13mm, 0.5mm pitch FCCSP

Texas Instruments
www.ti.com

DATASHEET

DATASHEET URLS:

Silicon Labs: https://www.silabs.com/documents/public/data-sheets/efm32pg23-datasheet.pdf

STMicroelectronics: https://www.st.com/resource/en/datasheet/stm32l051c6.pdf

Texas Instruments: https://www.ti.com/lit/ds/symlink/am625.pdf?ts=1732405727003&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FAM625

Embedded System Essentials

Holiday Hangover Hardware Hacking

Using a Raspberry Pi Pico to Hack an IP Camera

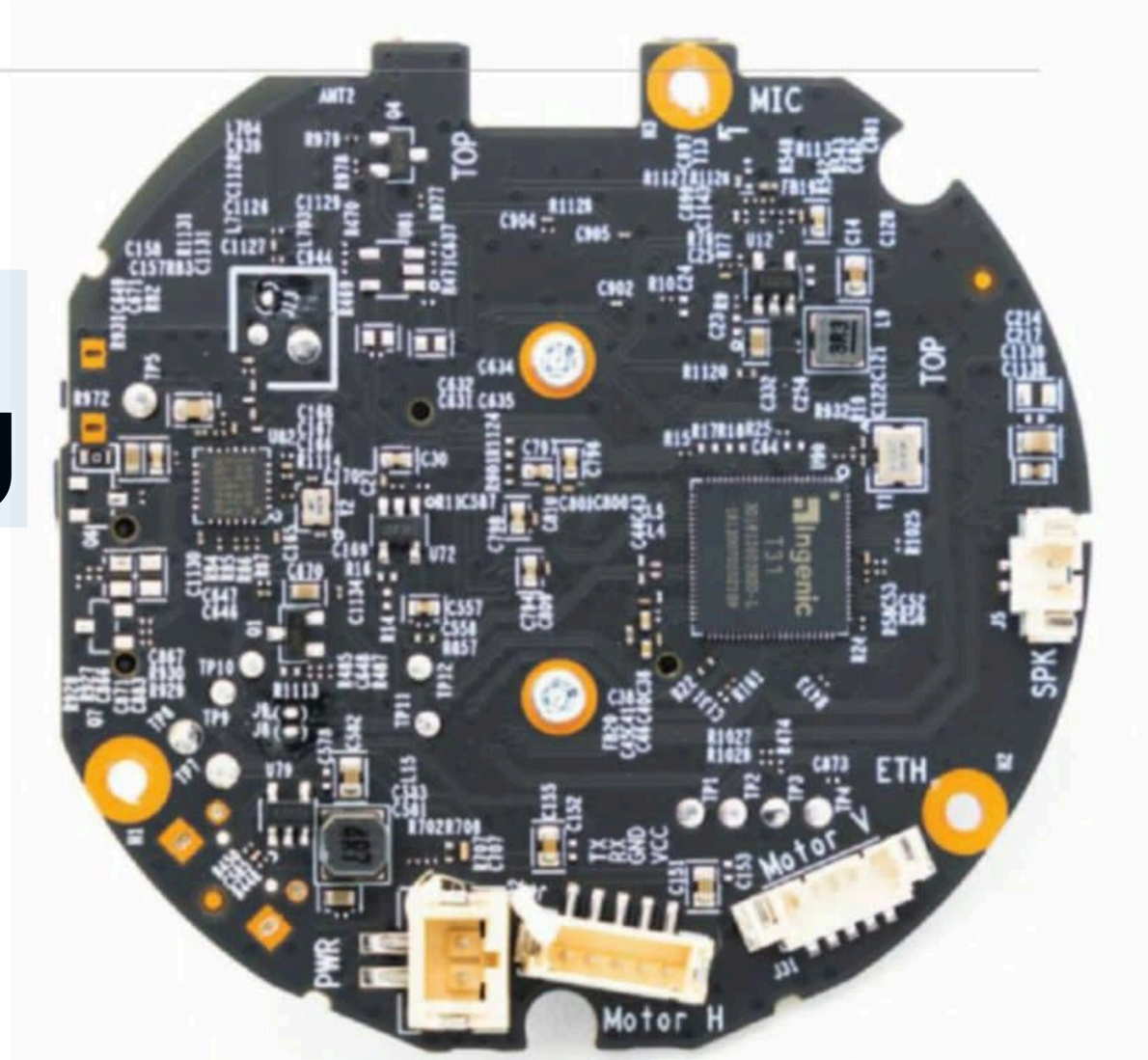


FIGURE 1
The main board of the Tapo C200 showing the T31 SoC.

By
Colin O'Flynn

COLUMNS

Having too much cheer during the holidays? In this month's article, Colin offers a diversion from the jolly season by urging developers to retreat to the basement to brush up on hardware hacking skills. He shows how a low-cost Raspberry Pi Pico and a TP-Link Tapo C200 smart IP camera could become the next automated bird deterrent or a home automation server.

The holidays are a great time to pick up some low-cost IoT hardware for practicing or developing your hardware-hacking skills. In this article, I'll go over some work on the TP-Link Tapo C200 smart Internet Protocol (IP) camera. I use this camera as the basis for several labs in my undergraduate course on cybersecurity at Dalhousie University. It's also a heavily attacked device. You'll find several nice websites and repositories that have similar work. In particular, I first used the work by DrmnSamoLiu on GitHub [1], but several other resources are also available [2]. Despite all the attacks, there is still lots of analysis left!

One alluring feature of this camera is its very low cost—it often appears in sales, reducing the cost further. And it features some interesting hardware, including a camera with pan-tilt capability, a microphone, and a speaker. It also runs Linux with Wi-Fi connectivity, so if you could run your own software on this device, you could turn it into anything from an automated bird deterrent to a local, miniature, home automation server.

To make this as accessible as possible, I'll use the Raspberry Pi Pico as the actual hardware-hacking tool. Onto this board we'll load several different firmware images to give us the various interfaces we need to work with the Tapo C200.

FIRST LOOKS

When looking at a device to analyze, one cheat is to use the FCC-ID that any wireless device is required to have. Looking up the FCC-ID of this device (2AXJ4C200V2) [3] will give you internal photos of the main board. I often use this to check if the device appears to have some internal headers that might make attacking it easier, before even buying it.

In this case, I already knew the device had lots of prior attacks, but if you are checking out a holiday deal on another IoT device, check the internal photos in the FCC filings to get an idea of what hardware is present. A photo I took of the main board is given in **Figure 1**. Of particular interest is the main System on Chip (SoC), which is an Ingenic T31. Once you find the datasheet for this device, it will tell you that the device is a MIPS32 architecture, highly integrated device, with DDR memory on-board and both video and image signal processors. On the other side of the board (not shown) is the SPI flash chip (XMC 25QH65C), which appears to be the only non-volatile memory on this device. The normal first order of business would be to see if we can dump this flash memory chip, which we'll do using the classic "clip on reader," before showing you how to do this over the serial port using U-Boot.

SPI FLASH DUMPING, ROUND 1

The normal way of dumping SPI flash would be to use a hardware interface with the flashrom tool. The first thing I did was program the Raspberry Pi Pico with the pico-serprog firmware made by hardware hacker Thomas Roth ("stacksmashing") [4]; specifically, I used a slightly extended version of it, from my companion repository for this article [5].

To use this tool, we simply attach the Raspberry Pi Pico, using either a SOIC-8 IC test clip or by connecting each of the SPI wires, as shown in **Figure 2**. The one caveat here is that we need to hold the main SoC in reset, to prevent it from attempting to access the SPI flash chip. In this case, we can find from the datasheet that SoC pin 7 is the reset pin. Looking carefully at the PCB, you'll find an open spot where a resistor or capacitor could be mounted on the trace connected to Pin 7 on the PCB. You can either use the pad to hold the reset line low with a wire, or just solder a blob onto the pins to hold the pin in reset.

Either way, once this is done, using flashrom should work

to dump the binary. See the companion repository [5] for the full description of the commands. With a binary in hand, we can do an initial analysis.

BINWALKING

The tool of choice for analyzing such firmware images is called binwalk. This open-source tool can automatically find interesting bits of data, and will automatically extract them if we run the simple command `binwalk -e spi.bin`, where `spi.bin` was the file we read out using flashrom.

Running the command will result in a number of interesting-sounding results from it. In particular, it has extracted the root filesystem to a new folder, "squashfs-root".

Looking at this filesystem, we can find two of the most interesting files for our IoT exploration, `/etc/passwd` and `/etc/shadow`. They contain a list of the users along with their hashed passwords. Copies of these files are given in **Listing 1** and **Listing 2,** respectively. The hashed passwords mean we'd need to run a password cracker. But we can compare multiple copies of the devices to confirm that the same password is used across all devices.

What is interesting on this device is that the shadow file in Listing 2 has an "x" in place of the hashed password. This prevents the login, since Linux is expecting a hashed password here. In fact, the hashed password probably should not have been left in /etc/passwd, but this bit of leftover information gave us a default password that was used at one point. Because the actual shadow file (which Linux is using) has this disabled, we know that we won't be able to log in directly, anyway.

But enough looking through files. Let's start talking to our device while it's operating.

SERIAL LINK ONLINE

Looking at the main board from Figure 1, the bottom right area of the board has some silkscreen markings reading TX/RX/GND/VCC, which suggest a serial port is present. I've separated out this section of the photo so you can get a closer look in **Figure 3**. In earlier versions of the hardware, you can solder header pins onto those pads and get a serial port. On the later hardware revisions, you need one more step. The zoomed-in view in Figure 3 shows a few 0402-size pads with no components mounted. These are labelled R1027 and R1028. The manufacturer has added some-do-not-mount (DNM) resistors to make your life a little harder.

You'll need to jumper those pads first, and then you'll find a serial port at 115200 baud rate. If you don't have a USB-Serial adapter, you can re-use the Raspberry Pi Pico

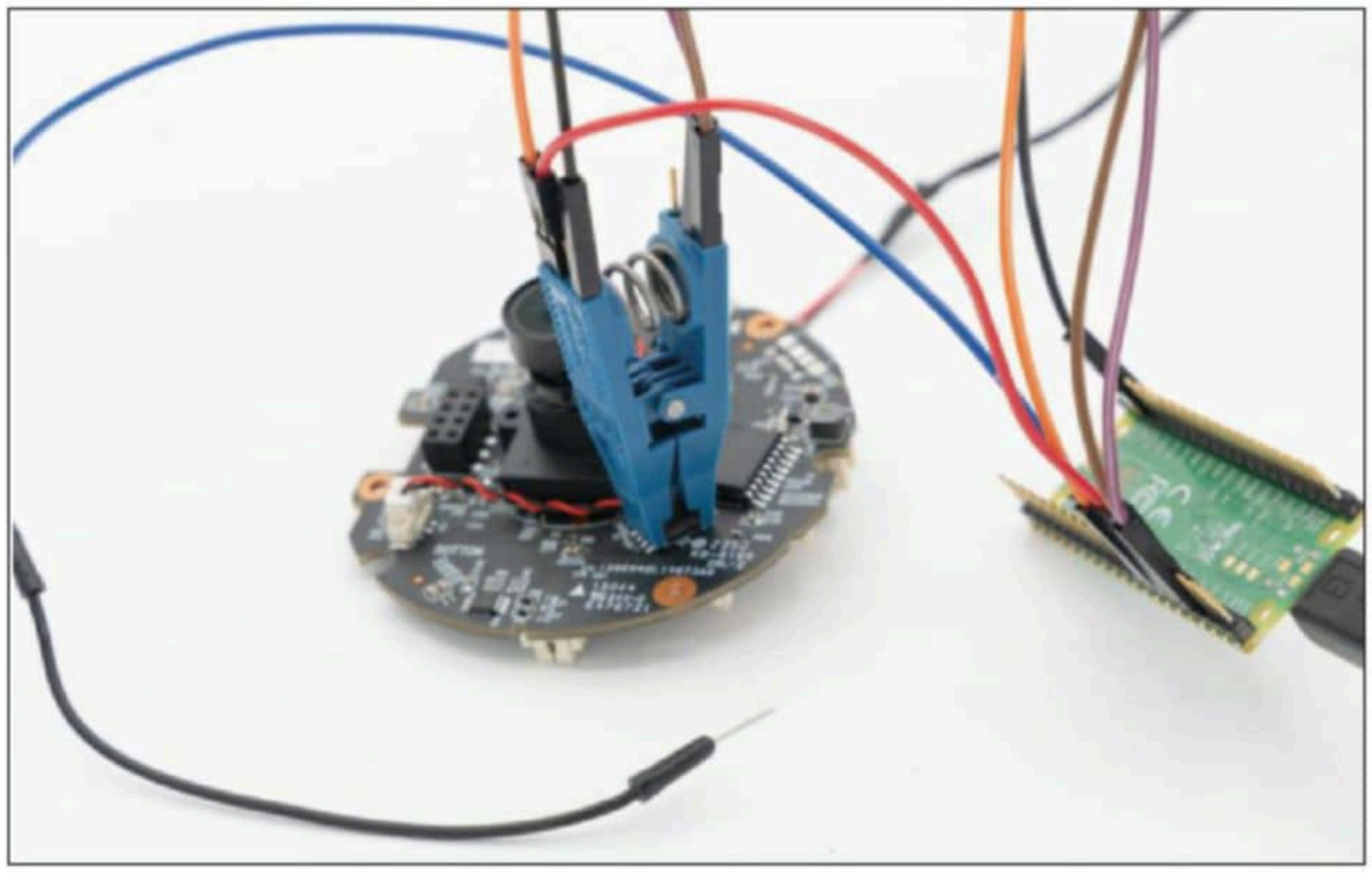


FIGURE 2
SPI Flash can be dumped with an SOIC8 clip, provided that the SoC is held in reset.

and reflash it with a USB-CDC firmware (see companion repository [5]). This firmware is the pico-uart-bridge project by Álvaro Fernández Rojas ("Noltari") [6]. As I promised this entire article uses the low-cost Raspberry Pi Pico for everything!

Once the system eventually boots, you get a login prompt, but if you enter the root user, you can get a response that the user is invalid (since it's been disabled). So we need to go a little further to get access to this system.

DAS U-BOOT, ROUND 1

The most common embedded Linux bootloader is called Das U-Boot (or just "U-Boot" for short). This bootloader has a powerful prompt that lets you access memory along with changing the Linux boot command line. One common attack is to modify the Linux command line to add an "init" argument that opens a shell and skips the login process entirely.

One of the first orders of business on an embedded Linux system will be to access the U-Boot console. During the boot process you can see the typical message that says

```
root:$1$ORLLCgGG$/OYVHO2yJhoAVshA3pdU40:0:0:root:/
root:/bin/ash
nobody:*:65534:65534:nobody:/var:/bin/false
admin:*:500:500:admin:/var:/bin/false
guest:*:500:500:guest:/var:/bin/false
ftp:*:55:55:ftp:/home/ftp:/bin/false
```

LISTING 1
The "passwd" file has the hashed root password, but in this case the shadow file (Listing 2) has this disabled.

```
root:x:0:0:99999:7:::
daemon:*:0:0:99999:7:::
ftp:*:0:0:99999:7:::
network:*:0:0:99999:7:::
nobody:*:0:0:99999:7:::
```

LISTING 2
Modern devices used the shadow file, which won't be readable during normal operation. The root user is disabled because there is no valid hash present.

"Autobooting in 1 seconds." (This is also shown in **Listing 3.**) In the old days of unsecured IoT devices, you could press Enter during this message to enter the boot console; but these days, manufacturers try to disable this attack.

Often, however, we can force the bootloader entry by shorting the SPI flash pins as the correct time, as shown in **Figure 4**. This type of attack is called PIN2PWN. The timing here is critical—we need to leave the SPI flash working while the U-Boot bootloader loads, but then short the pins when it tries to load the Linux kernel. If we're successful, we'll see a message like Listing 3, which is the U-Boot console. You might have to experiment with the timing to get this working, or just jump ahead to the second method.

As a special hint: this device actually has two U-Boot consoles. If you watch the normal boot process, you'll notice two sequences that say "Autobooting in 1 seconds," and we actually want to target the second U-Boot console.

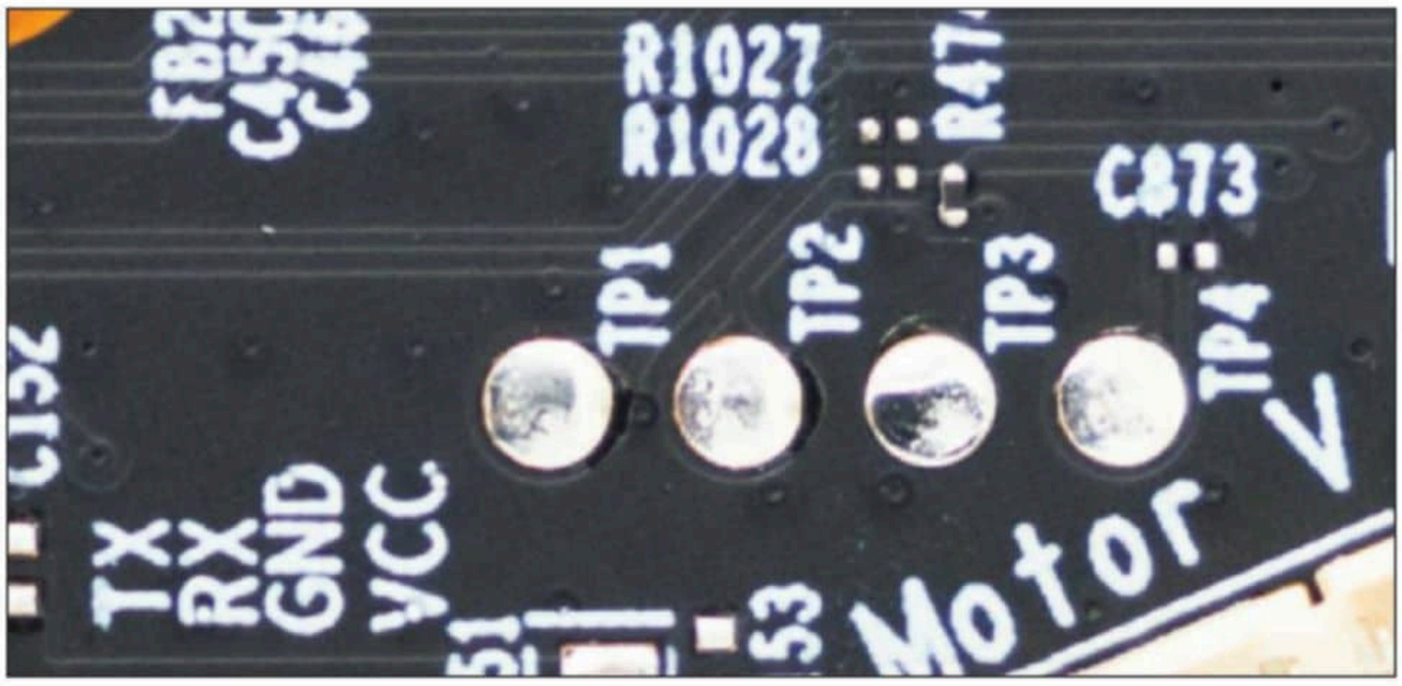


FIGURE 3
You'll need to either mount R1027 and R1028, or just short those pads with solder to activate the serial port.

```
Now running in RAM - U-Boot at: 83fd0000
MMC:   msc: 0
the manufacturer 20
SF: Detected XM25QH64C

In:    serial
Out:   serial
Err:   serial
GPIO : 67894000 -> 67894000
the manufacturer 20
SF: Detected XM25QH64C

--->probe spend 4 ms
SF: 8388608 bytes @ 0x0 Read: OK
--->read spend 2687 ms
------Firmware check pass!-----
Autobooting in 1 seconds
isvp_t31#
```

LISTING 3
Watching the boot process shows you where to interrupt the boot to drop to the U-Boot console, interrupting it either using PIN2PWN or the serial password.

DAS U-BOOT, ROUND 2

Besides using PIN2PWN, how does the manufacturer enter U-Boot during development? It turns out there is actually a password for the bootloader. This was listed on the prior work webpage, and was discovered as part of analyzing the open-source code released for GPL compliance.

This means you can also enter the U-Boot console by simply typing `slp` over the serial port during the autoboot phase. Again, you ideally want to enter the second U-Boot instance. If this works, you'll get the same console that is the final line of Listing 3.

SPI FLASH DUMPING, ROUND 2

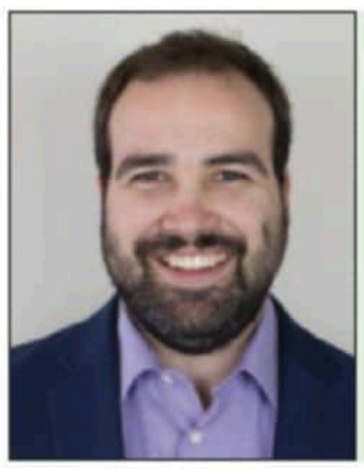
If you didn't have an SOIC-8 clip, we can use the U-Boot console to access the SPI flash. The commands in **Listing 4** first detect the flash (sf probe), followed by copying a section of it to internal memory (sf read). The final command dumps 256 bytes of the internal memory. We can transfer off this internal memory copy using a simple script. The addresses of the flash memory and the internal memory were printed during the boot process, or you could look at the commands run on boot normally to understand the address space.

The companion repository [5] includes an example Python script to dump the entire flash automatically using this interface. There are more advanced U-Boot features that we can use to transfer files, but this shows that a simple console provides everything we need to read the entire filesystem out, with almost no additional hardware.

But reading the filesystem out isn't the most interesting thing we can do. The grand finale is to log into the system "live," so we can explore it on the hardware, itself.

LINUX PASSWORD BYPASS

If you issue the command `printenv` in the U-Boot console, you'll get some environmental



ABOUT THE AUTHOR

Colin O'Flynn (colin@oflynn.com) has been building and breaking electronic devices for many years. He is CTO of NewAE Technology based in Halifax, NS, Canada and is currently an assistant professor at Dalhousie University. Some of his work is posted on his website at www.colinoflynn.com.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] to [6] as marked in the article can be found there.

COLUMNS

variables; a subset of them is shown in **Listing 5**. In particular `bootargs` is the Linux boot arguments, and `bootcmd` is the command run to boot the kernel. We can change `bootargs` by running the command in **Listing 6**. This will change Linux, so it boots into a basic shell, instead of running the normal initialization commands.

If your `bootcmd` differs from Listing 5, you might be in the wrong U-Boot instance. The other U-Boot instance seems to have an invalid boot argument.

Once you're in this console, you can explore the system "live." Because we interrupted the boot process (replacing the correct `init` call with our shell), many systems will be broken. You can improve this by first running the command `/etc/preinit`, which was the command we skipped. You might then want to run some of the commands in `/etc/init.d` for example, but you'll likely find many things broken, since this just gave us a basic console. To get persistent access, we'd need to rewrite the root password and save it back to the SPI flash, and enable the root user. You could also turn on a more convenient network-based interface such as an SSH server. All of this will be a topic for future exploration!

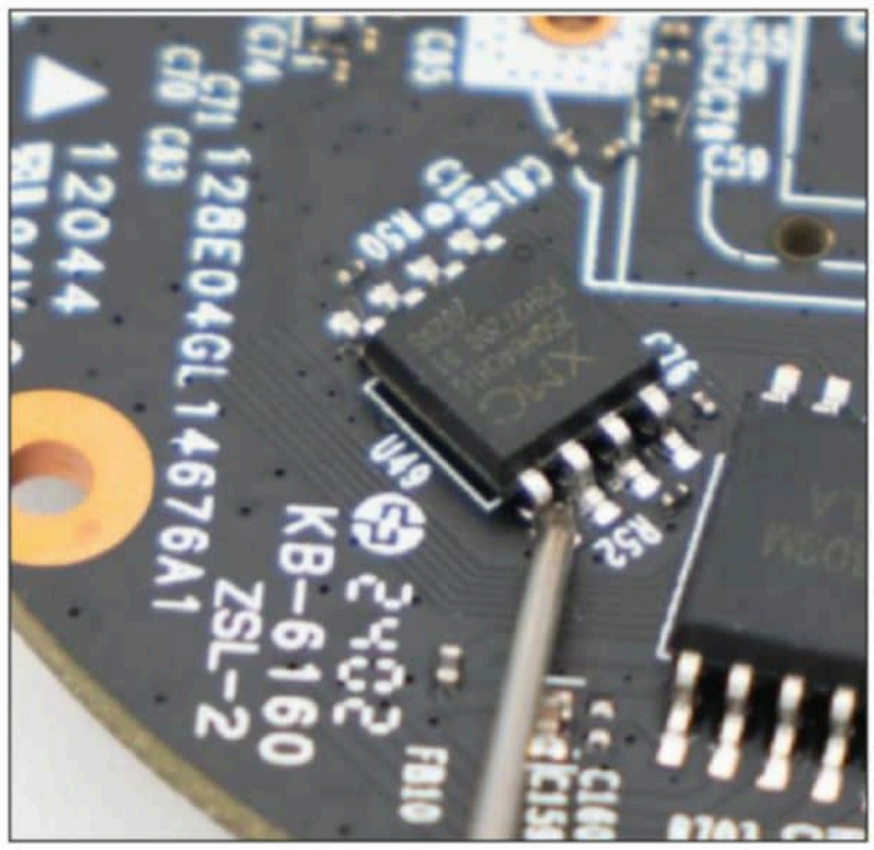


FIGURE 4
PIN2PWN causes an invalid SPI data response that causes the bootloader to drop to a recovery prompt, requiring just something to short two of the SPI data pins.

FUTURE WORK

In this article, I demonstrated how you can get into hardware hacking with low-cost tools. As of this writing, the Raspberry Pi Pico is $4 from Sparkfun. You'll spend more buying the target and even the jumper wires than you will on the Pico, itself.

The C200 IP camera is an interesting target, because a robust community of hardware hackers is looking at it, and the device itself has a lot of capabilities. With a little more effort, you can run your own applications on it. But even if you don't plan on doing that, it is a nice project to work on during the long winter nights.

Be sure to check out my companion GitHub repository [5], where I've posted some of the scripts developed during this work. It also includes links to several other hardware hackers who have explored the device, including a graduate student here at Dalhousie who found how you can blink the lights on your device.

```
isvp_t31# sf probe
the manufacturer 20
SF: Detected XM25QH64C

isvp_t31# sf read 0x800000 0 0x100000
SF: 1048576 bytes @ 0x0 Read: OK

isvp_t31# md.b 0x800000 256
00800000: 06 05 04 03 02 55 aa 55 aa 0c 00 00 98 40 00 00    .....U.U.....@..
00800010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
...
```

LISTING 4
Using the U-Boot console allows you to dump the SPI flash contents directly.

```
isvp_t31# printenv
baudrate=115200
bootargs=console=ttyS1,115200n8 mem=45M@0x0 rmem=19M@0x2d00000 root=/dev/mtdblock6
rootfstype=squashfs spdev=/dev/mtdblock7 noinitrd init=/etc/preinit
bootcmd=sf probe;sf read 0x80700000 0x80200 0x175000; bootm 0x80700000
bootdelay=1
```

LISTING 5
The environmental variables include the Linux boot arguments.

```
setenv bootargs console=ttyS1,115200n8 mem=42M@0x0 rmem=22M@0x2a00000
root=/dev/mtdblock6 rootfstype=squashfs spdev=/dev/mtdblock7 noinitrd
init=/bin/sh
```

LISTING 6
We can re-write the boot arguments to boot to a simple shell.

PICKING UP MIXED SIGNALS

Using Amazon Alexa to Control Custom IoT Gadgets

Part 2: Emulation and Spoken Alarms

By
Brian Millier

In part two of his article, Brian describes integrating custom IoT gadgets with Amazon Echo using emulation to receive spoken alarms. In part one, he used emulation and Arduino Cloud services as a middleman.

COLUMNS

In Part 1 of this article, I described two different methods of designing a custom IoT device that could easily be integrated with the Amazon Echo smart devices by making use of the Alexa speech recognition feature. One method involved the emulation of a commercial Alexa-compatible device, the Philip's Hue smart light bulbs. The second method used the Arduino Cloud service as a "middleman" between the custom IoT device and the Amazon Alexa cloud server.

In this article, I'll describe another custom IoT device that uses the emulation method. To conclude the series, I'll describe an IoT device that can notify your Echo device of an alarm condition, allowing you to receive spoken warnings, and notifications to your cellphone or tablet.

TRICKY THERMOSTAT

I became very interested in electronics when I was about 10 years old. I don't recall how I justified it, but, as a teenager, I remember taking a thermostat off the wall at home, to see how it worked. **Figure 1** shows one of these old round Honeywell thermostats, with its cover removed.

How it operated was fairly obvious: There was a bimetallic strip wound in a spiral shape. At one end of it was a glass enclosed mercury switch, oriented horizontally. Turning the temperature-setting dial must have adjusted the other end of the spiral, I believe. As the room temperature changed, the spiral would wind/unwind and the mercury switch would tilt, opening and closing the switch contacts. I also recall seeing another small dial that was not visible from the outside but can be seen in the foreground of Figure 1. Later, I found out this was called an anticipator adjustment and it existed to counteract the tendency for the room to continue to heat up somewhat, even after the furnace's burner was shut off by the thermostat. This dial was marked off in Amp units, I believe likely to match the current draw of the relay that the thermostat switched. This was 50+ years ago, but I think that the anticipator dial was associated with a small resistive heater. This could "trick" the bimetallic spiral into thinking that the room was warmer than it actually was, thus performing the anticipator function, mentioned earlier. If this heater were switched on/off by the thermostat's contacts, it would

FIGURE 1
This is a picture of the old Honeywell thermostat like the one I investigated as a teenager. (Image source: Wikipedia).

also provide what is called hysteresis in the instrumentation field. Hysteresis is needed to eliminate rapid thermostat switching, when the temperature fluctuates slightly around the setpoint. If you watched the mercury switch, it was obvious that it flipped from on to off with just the slightest change in position of the bimetallic spiral. It isn't good for the furnace burner to be cycled on and off rapidly- thus the need for this hysteresis.

Recalling this experience in my youth, it occurred to me that a programmed "set-back" function could be added to a modern thermostat in my home. Where the old Honeywell thermostats switched 24 VAC, at only the low current needed to operate the furnace control relay, I have electric heat and such thermostats must switch 240VAC at up to 20A.

The electric heat thermostat, in the room that I was interested in, contained a bellows and an adjustment screw coupled to the temperature dial. The high-current, 240VAC power switch that was activated by this bellows, was totally enclosed in the plastic rear section of the thermostat (fastened to a junction box in the wall). It occurred to me that I could glue some low-value resistors inside the decorative front cover surrounding the bellows assembly to act as a small heater.

During the time that I wanted the thermostat to set back (for example, lower the room temperature), I would only have to supply power to those resistors mounted inside the front cover. When they heated up, they would fool the thermostat into thinking the room was warmer than it actually was. I chose the value of the resistors empirically and initially adjusted the voltage feeding them, using a DC bench supply. With power applied to the resistors, I wanted to observe the thermometer (contained in the thermostat), reading 5°C hotter than the actual room temperature. This would implement a roughly 5°C set-back when the resistors were powered up. My aim was to be able to use a small wall adapter to supply this power. I had plenty of them around between 3V and 15V. It turned out that a 4.5V 1.5A adapter I had on hand was perfect for the job. I used four series-connected 1.0Ω 1W resistors in the thermostat itself (as the heater) and an additional external 1.0Ω resistor, in series with the 4.5V wall adapter. **Figure 2** shows the thermostat rear assembly and **Figure 3** shows the front panel, with the four resistors mounted.

To control the power to this wall adapter, I wanted to use Alexa to manage the daily temperature set-back periods. In this room, I was already using an Alexa-compatible TP-Link KP2000 Smart duplex wall receptacle-for another purpose. This product allows individual remote control of each of its two receptacles, using Alexa. So, I just plugged the 4.5-V wall adapter for my set-back thermostat, into the KP2000's unused socket. This TP-link receptacle requires you to add the Kasa Skill to your Alexa profile. To control the

FIGURE 2
This is a modern thermostat meant to control baseboard electric heat. The high current switch is completely enclosed in the brown plastic rear assembly. The bellows which activates it is visible in the center..

FIGURE 3
This shows the four 1W resistors that I am using as a heater to "trick" the thermostat into thinking the room is somewhat warmer than it really is.

temperature set-back periods, it was simply a matter of adding a new Routine to my profile and setting the on/off times appropriately. When I know in advance that I am going to be using this room (a music studio in my basement), I can quickly turn off the set-back function, with a voice command or from my cellphone's Alexa app, 10-15 minutes before I plan to use the room.

ALEXA TRIGGERS

The last category of gadgets that I tackled were ones that would send a message to my Echo devices, acting as a trigger for some action that I wanted my Echo devices to perform. These messages could occur at a random time: Think of an alarm for example, as opposed to an IoT device that is periodically polled by the Echo device. Such an alarm message can be programmed in the routines section of the Alexa app, to perform many different functions. A few such functions, that I thought would be useful, are as follows:

- Send a notification to my cellphone and iPad tablet.
- Make a custom, meaningful voice announcement on one or all my Echo devices.
- Turn on a light bulb or other electrical device.
- Use Alexa routines can generate social media posts as well.

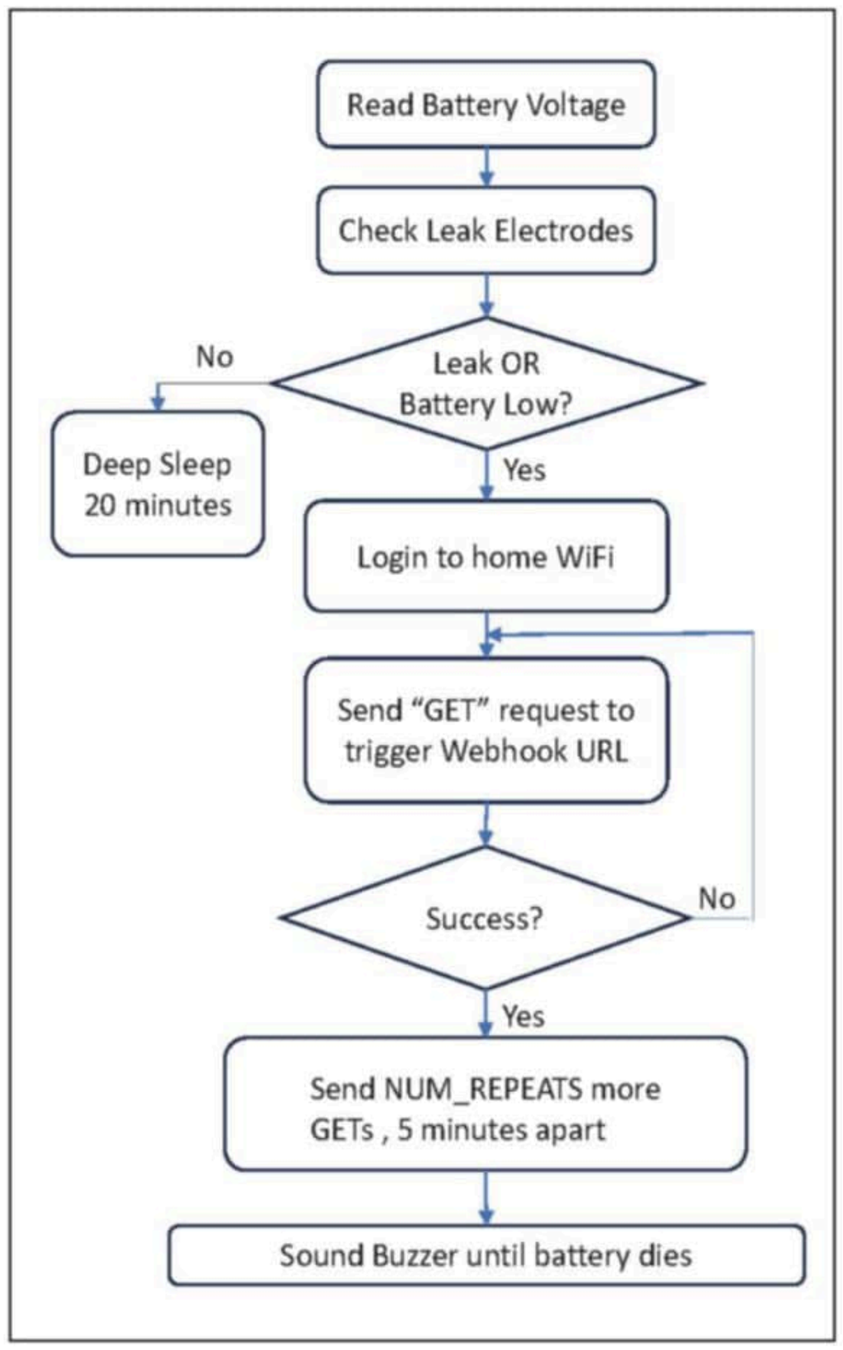


FIGURE 4
This is a block diagram of the program running the water leak detector. It sends a message to the mkZense server via a webhook. That gets passed on to the Amazon Alexa server and ultimately sent to my Echo devices.

Except for the Arduino Cloud-based IoT device example in Part 1, the device examples communicated with the Echo device directly. That is, the custom IoT device implemented a webserver, configured to emulate a commercial IoT product, i.e., a Philips Hue smart bulb. The Echo needed the "Skill" for that product loaded and enabled. Then it would "discover" these custom IoT devices over my home's Wi-Fi network.

In the case of Alexa triggers, the process is somewhat more complicated. To start with, Amazon has wisely decided that it would not let its Echo devices respond directly to incoming messages from any random device that you might connect to your WiFi network. You can readily see that this could lead to many bad outcomes, particularly if outside acters managed to gain access to your home WiFi network.

For Alexa triggers to work, the trigger message must be sent to the Amazon Alexa server directly. For this message to be properly routed to your specific Echo device, the trigger message must include a "key" of some sort, to tell the server which specific Echo device to send the trigger message to. To generate this key, your Amazon account information must be provided to prevent outside actors from being able to access your Echo devices maliciously.

Furthermore, the Amazon server must be able to authenticate the source of this incoming trigger message and not just the validity of the message itself. In other words, the server or device sending the trigger message must be able to complete an authentication routine using credentials, recognized in advance, by Amazon. It's certainly possible for an individual with competency in security protocols to design an IoT device that meets the stringent criteria necessary to access Amazon's Alexa API. However, for the average DIY individual, it's much easier to find a server, run by a third-party, that provides a "middleman" service between that individual's custom IoT device and the Amazon Alexa server.

Since this "middleman" server must be able to link the trigger messages that you are sending to your personal Echo device, it follows that your Amazon account information must be involved in some way. The fact that you own and use Echo devices means that you trust Amazon with personal information, including credit card information. However, you might be disinclined to give an unfamiliar company, acting as the "middleman", access to your Amazon account information. I know I initially was so disinclined.

Amazon solved this trust problem by "linking" your Amazon account and that of the third-party middleman, in the following

way: When you set up an on-line account with the third-party "middleman", at some point in that procedure, you are re-directed to Amazon's server. Once you successfully login to the Amazon server, Amazon will then send back to the "middleman" (the identity of which Amazon has previously established), whatever authorization keys are needed. This authorization will:

- Establish the identity of the "middleman" server.
- Identify your specific Amazon account as the destination for the trigger message.
- Specify an IoT device from which the message originates.

When I searched for "Alexa Trigger" skills from the Amazon Alexa cellphone app, I came up with a few possibilities. The first ones listed were:

- IFTTT Trigger by mkZense.
- Virtual Routine Trigger.
- URL Routine Trigger.
- Webhook Routine Trigger (similar to the mkZense offering, but based in Germany).

I've had previous experience using the IFTTT cloud service with earlier IoT devices I had built. That experience wasn't so positive: features that I used, that were initially free, were later migrated over to a paid, monthly subscription plan. Nevertheless, I first investigated the IFTTT Alexa Trigger skill, as it looked promising. The company is mkZense, and they allow you to open a free account which allows you to define a couple of triggers. As mentioned earlier, setting up this free mkZense account is done using the Alexa cellphone app and adding the IFTTT Trigger Skill to your Alexa profile. This signup initially takes you to mkZense's website but also involves being redirected back to the Amazon site, where you must login with your Amazon account username and password. This links the two accounts. As part of this procedure, you will be prompted to enter descriptive names for the number of triggers you want to set up.

With the free mkZense account set up, you next go to the IFTTT website and set up an applet that integrates the MkZense service with Amazon's Alexa service. The mkZense service has videos describing how to do this and claim that you can create an IFTTT applet without a paid ITFFF subscription. While this may have been possible when they posted the video, this is no longer the case. You don't discover that you need a paid account until you reach the end of the IFTTT applet configuration. This was disappointing to say the least, but I noticed that mkZense has a paid subscription plan that bypasses the IFTTT service completely and allows for an unlimited number of triggers. Since this cost only $5 per year, and can be paid for using Amazon Pay, I decided to sign up and proceed this way. Note that if you initially add or enable the free mkZense Skill, you must disable this skill in the Alexa app and start over again, selecting the paid subscription. Once you have subscribed, you will receive an email at the email address associated with your Amazon account. This email will contain a Webhook access token, which is a long string containing hex characters.

What is a Webhook? A Webhook is a unique internet URL composed of the URL of an operational webserver, combined with an access token which points to a specific "webpage" on that webserver. When you access that specific URL, the server recognizes that access and performs some relevant action. In this instance the server is run by mkZense and it will send the proper trigger message to the Amazon Alexa server. The Alexa server will in turn pass this message on to your Echo device. Following is an example of the mkzense Webhook URL that you might get in that email:

https://mkzense.com/webhook/alexa/
7a09384635ef1f37c4d671b9c7724151901
957/TRIGGER-NAME

You can see the mkZense webserver address followed by "webhook/alexa/" and an access token. At the end is the word TRIGGER-NAME. You must replace this with one of the

```
while (repeats < NUM_REPEATS)  {
  Serial.println("connecting to mkzense webhook server");
  http.begin(client,"http://mkzense.com/webhook/alexa/7a09384675efle37c4c
  int httpCode = http.GET();
  String payload= http.getString();
  http.end();
  int found;
  found= payload.indexOf("SUCCESS");
  if (found >0) {
   Serial.println("connected to mkzense successfully");
   repeats++;
   delay(3000000); // wait for 5 mins before next transmission of message
 }

else {
   Serial.println("failed to reach mkzense successfully");
   retries++;
   // break out of while loop after 5 unsuccessful attempts
   if (retries > NUM_RETRIES) repeats=NUM_REPEATS;
   delay(60000);  //wait a minute before trying again
}
}
```

LISTING 1
This is a snippet of code that performs a GET request to the URL of the mkZense webhook, which acts upon that webhook access by forwarding the alarm message to the Amazon Alexa server.

actual trigger names that you supplied. In case you have forgotten them, your registered trigger names are listed at the end of this email message.

You must next set up a Routine using the Amazon Alexa app. When setting up this routine, for the WHEN condition, you would select the Smart Home section, where you should see entries for all the triggers that you defined when you added the "IFTTT Trigger" Skill to your Alexa profile. Select the applicable trigger device. Then, in the "ALEXA WILL" section you can choose such actions as Notification, Announce, etc.

At this point you can test out the proper operation of this webhook by cutting/pasting the URL into your PC's web browser's URL address field. When you access this URL, your Echo device should perform whatever task you have defined in the Routine that you set up for that trigger. Also, your web browser should display something that looks like the following:

"SUCCESS::north-america:YOUR_EMAIL_ADDRESS.com:TRIGGER-NAME"

The message returned is formatted in JSON file format and will be displayed in JSON format, if your browser supports displaying that format.

For your IoT gadget to be able to trigger the desired Alexa Routine, it must be capable of performing a "GET" request to the trigger's URL. In my case, I was adapting water leak detectors, that I had built in the past, to connect to Alexa. Those detectors were built using Espressif's ESP8266 MCU and I program these devices using the Arduino IDE, with the ESP8266 board support package added.

A block diagram of the program flow needed to implement this is shown in **Figure 4**. I'll discuss circuitry later, but for now, if either a water leak is detected or the battery voltage is found to be getting low, the ESP8266 will log in to my home WiFi access point (repeatedly if not initially successful). It will then send the "GET" request to the Alexa trigger URL shown earlier (but edited-for my privacy). To do so requires the "ESP8266HTTPClient" library, which can be obtained using the Arduino Library Manager. **Listing 1** shows the snippet of code needed to send this request, as well as the code which checks for the success of this procedure.

Note that in the third line, the long trigger URL is not completely shown, but the full URL, as described earlier must be used. The line containing httpCode= http.get(); performs the actual GET request and returns the httpCode variable. The http.getString() routine fetches the JSON result string, sent

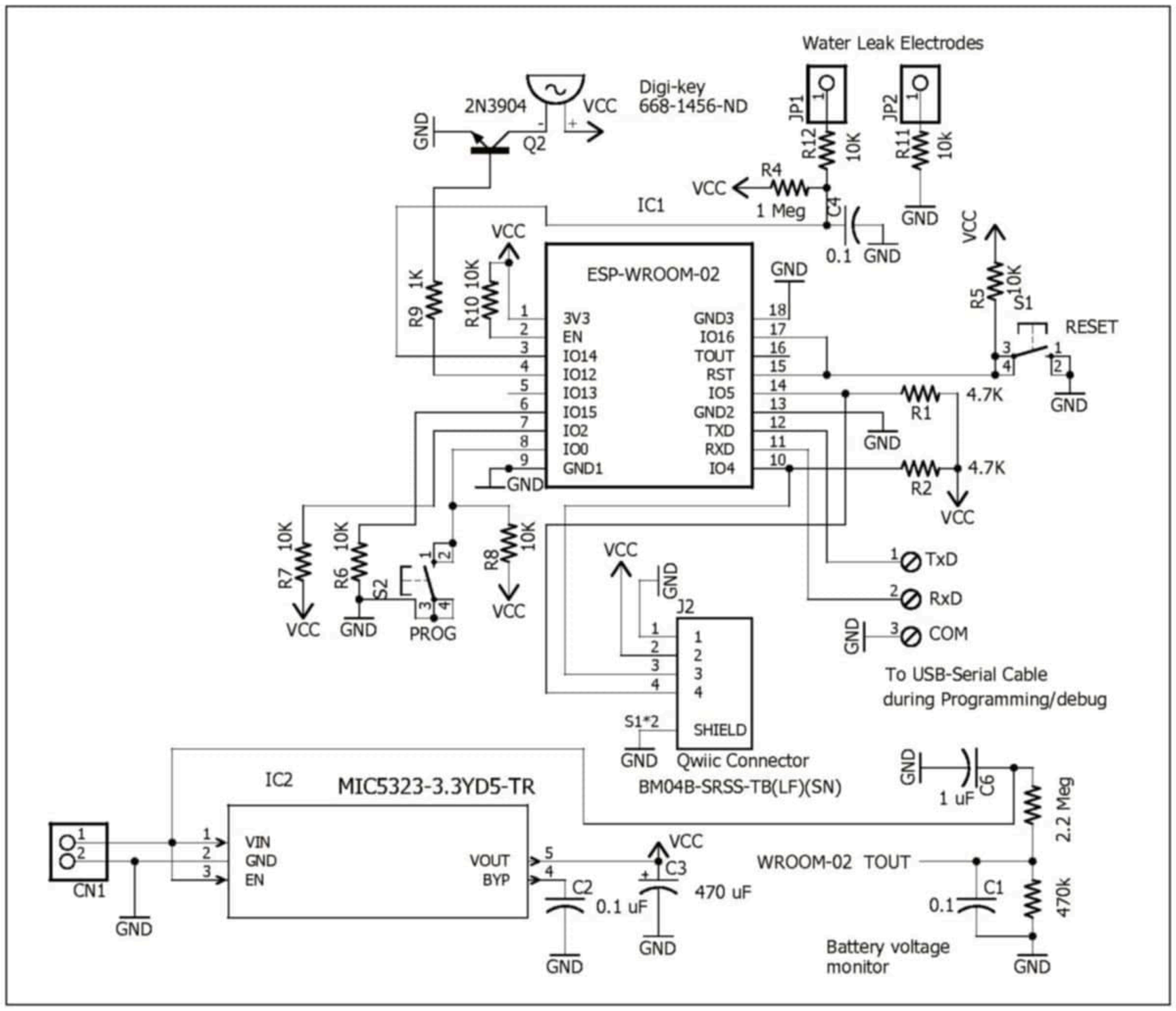


FIGURE 5
This is the schematic diagram of the water leak detector, which sends its alarm signal to my Echo devices. The Echo devices can be configured to do several things when triggered by this alarm signal, including sending notifications to both my cellphone and iPad.

COLUMNS

by the host webserver. I use the indexOf function to search for the "SUCCESS" string in the JSON response and increment the repeats variable if it's found. If successful, I will send this GET request NUM_REPEATS more times, at 5-minute intervals, to maximize the chance that I will hear Alexa when she announces the water leak. In any case, several notifications will be sent to my phone or iPad.

If unsuccessful at reaching the mkZense server after NUM_RETRIES attempts, the loop will be exited, and the on-board buzzer will sound until the battery dies. This beeping will also occur if the mkZense server is reached and the Alexa skill is triggered on the Echo device(s).

LEAK ALARM CIRCUIT

Figure 5 is the schematic of the water leak alarm. Battery power is applied at CN1 and can either be a 3.7-V LiPo cell or 3 AAA cells in series. The battery voltage is regulated down to 3.3-V by a MIC5323-3.3 LDO regulator, which was chosen for its low quiescent current value. The water leak electrodes are two 1" wires spaced about ¼" apart. One of them is connected to Vcc through a 1 Meg resistor, the other to GND, and each electrode has a 10k resistor in series with it to protect the ESP8266 against static discharge. When the electrodes sense water, the voltage across the electrodes will drop from Vcc down to less than the Vcc /2 logic threshold of the ESP8266's IO14 digital input pin. The ESP8266 spends most of its time in deep-sleep mode. Every 20 minutes, it wakes up and checks the leak electrodes. It also measures the battery by monitoring the voltage divider consisting of a 2.2 Meg and 470k resistors, which are connected directly across the battery. While not used in this project, a Qwiic connector is connected to pull-up resistors and IO4, IO5- to allow an I²C sensor (humidity, temperature etc.) to be attached. Port IO12 drives a 2N3904 transistor to drive a small piezo beeper. This beeper must contain its own driver- the Digi-key 668-1456-ND for example. The ESP8266's Rx and Tx UART pins are connected to terminals to allow connection to a PC, using a USB-UART cable. This is needed to program the ESP8266's flash and is handy for debugging purposes.

ABOUT THE AUTHOR

Brian Millier runs Computer Interface Consultants. He was an instrumentation engineer in the Department of Chemistry at Dalhousie University (Halifax, NS, Canada) for 30 years.

COLUMNS

SIGLENT® And Saelig UNIQUE ELECTRONICS

Present the

From $339

SDS800X HD
HD Digital Storage Oscilloscopes

- 70 MHz, 100 MHz, 200 MHz bandwidth models
- 12-bit high resolution
- Real-time sampling rate up to 2 GSa/s
- Memory depth up to 50/100 Mpts
- Waveform capture rate up to 500,000 wfm/s (Sequence mode)
- 7-inch touch screen, support for external mouse and keyboard

• www.saelig.com • 585-385-1750 •

FIGURE 6
This is a screen capture of the PC program that acts as the user interface to my Joulescope power monitor. I measure a deep-sleep current of 92 uA and an average operating current of 29.7 mA for the 1 second interval when the program is actually running.

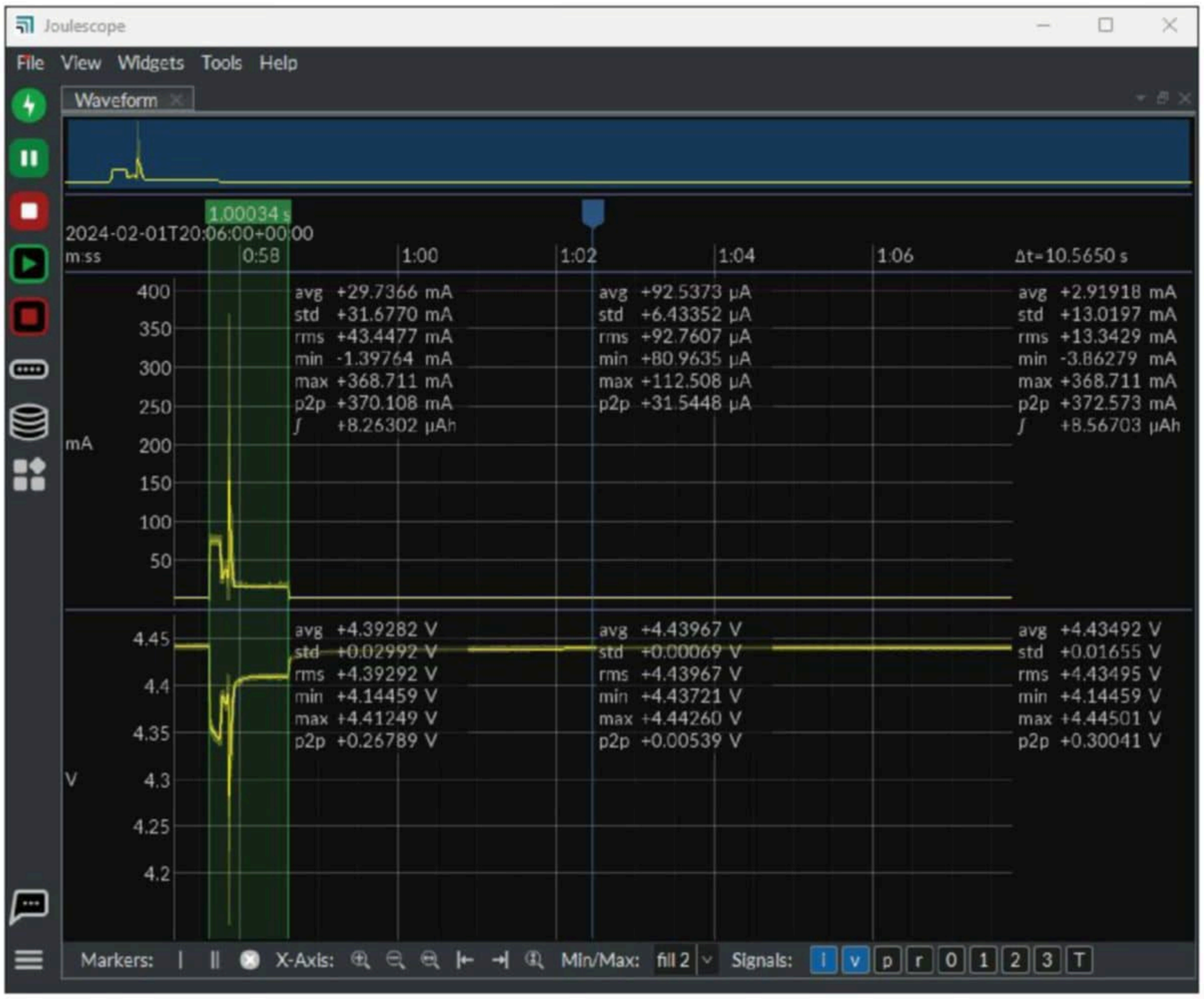


Figure 6 is a screenshot of my computer screen, taken while measuring the unit's power consumption using my Joulescope power monitor. During the 20 minute deep-sleep interval, the unit draws 92uA, comprised of the ESP8266 deep-sleep current, the quiescent current of the MIC5323-3.3 LDO, and some passive resistance loads. The ESP8266 program takes one second to monitor the battery voltage and check for leaks, and if both are OK, it immediately returns to a deep-sleep state. During this interval, it draws an average of 29.7mA. Using these figures, I estimate the battery life to be about one year, assuming a 2000 mAh LiPo battery.

CONCLUSION

I hope that you found this two-part article helpful as an introduction to integrating your own custom IoT devices to the Echo/Alexa smart home system. I suspect that there are many other ways of doing this, using either an emulation of a commercial Alexa-compatible device, or some other cloud service that is already linked to the Amazon Alexa server. If you are more ambitious and skilled, you can also obtain your own account on Amazon's server, figure out how to do the necessary authentication procedure allowing access to the Alexa API, and develop your own IoT gadget that isn't an emulation or needs a third-party server to act as the "middleman".

I briefly "toyed" with using an ESP32 MCU (as well as a Particle Photon 2) with the Edge Impulse AI cloud platform for "on-chip" voice recognition purposes. That seems to involve a lot of effort to achieve even very limited word recognition: granted, it's completely self-contained on the MCU itself. Unless you have access to a large library of spoken word samples, from a variety of different people, it's going to be speaker dependent. In contrast, the speech recognition capability of Alexa is quite remarkable, in my opinion. The extremely short time that it takes for your verbal commands to be sent to the server, recognized and the appropriate action taken, is also quite remarkable.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials

Resources

Arduino | https://www.arduino.cc

DFRobot | https://www.dfrobot.com

Espressif | https://www.espressif.com

mkZense | https://mkzense.com

COLUMNS

By Kirsten Campbell **PRODUCT NEWS**

New TI MCUs Enable Edge AI and Industry-Leading Real-Time Control to Advance Efficiency, Safety, and Sustainability

Texas Instruments (TI) introduced two new series of real-time microcontrollers that deliver advancements to help engineers achieve more intelligent and secure processing in automotive and industrial applications. TI's TMS320F28P55x series of C2000 MCUs is the industry's first portfolio of real-time microcontrollers with an integrated neural processing unit (NPU), enabling fault detections with high accuracy and low latency. The F29H85x series is built on TI's new 64-bit C29 digital signal processor core and offers an advanced architecture with integrated safety and security features

"Engineers increasingly strive to design automotive and industrial applications that are more energy efficient and can make faster decisions, driving a need for more scalable processing power, expanded memory, and on-chip safety and security enablers," said Amichai Ron, senior vice president, TI Embedded Processing. "For decades, our C2000 MCUs have been enabling manufacturers to measure and efficiently process real-world data. Today's introduction of enhanced real-time performance and edge AI in our C2000 portfolio will empower engineers to solve more complex problems and reach higher levels of system efficiency, safety and sustainability."

Engineers today are challenged to design systems that can make accurate, intelligent decisions in real time to perform functions such as arc fault detection in solar and energy storage systems and motor-bearing fault detection for predictive maintenance. As the industry's first real-time MCUs with an integrated neural processing unit, TI's new C2000 TMS320F28P55x series meets both of these challenges by ensuring consistent performance with real-time processing.

The NPU in the TMS320F28P55x series offloads the execution of the neural network model from the main CPU, achieving five to 10 times lower latency than software implementations to enable faster, more accurate decision-making. Further, the model that runs on the integrated NPU learns and adapts to different environments through training, helping systems achieve greater than 99% fault detection accuracy to enable more informed decision-making at the edge. TI's complete AI toolchain, which includes models that are optimized and tested for specific applications, helps engineers with any level of experience easily complete the AI model development process.

Texas Instruments | www.ti.com

New IX Industrial Ethernet Cables with Angled Connectors Are Compact but Rugged

L-com, an Infinite Electronics brand and a supplier of wired and wireless connectivity products, has just added a new line of IX industrial Ethernet cables with angled connectors. The Cat 6a assemblies solve tight cable routing issues in confined spaces with right-angle up or down options. They are ideal for factory and other industrial settings, especially security and machine vision camera applications.

The new cable line is rated IX industrial Type A for 10-gigabit Ethernet in harsh environments requiring compact, robust connectors. All use 26 AWG wire and are SF/UTP or F/UTP shielded for protection against electromagnetic interference (EMI) and radio-frequency interference (RFI). They are 100% PoE capable and compatible.

The newly released line includes versions for outdoor-and-high-flex-industrial, CMX low smoke zero halogen (LSZH), and CMR PVC economy jacket types. Each version comes in lengths of 1 meter or 3 meters.

The outdoor-and-high-flex-industrial assemblies are offered in three configurations: a down-angle IX A plug on one end and options of an IX A plug, RJ-45 jack, or RJ-45 plug on the other end. They are protected by a thermoplastic elastomer (TPE) outer jacket for flexibility, durability, chemical resistance and temperature tolerance.

The LSZH cable assemblies feature six configurations: down-angle IX A plug to IX A plug, to RJ-45 jack or to RJ-45 plug; and up-angle IX A plug to IX A plug, to RJ-45 jack, or to RJ-45 plug. The flame-retardant jacket is rated CMX LSZH for low smoke emission, zero halogens and reduced toxicity in the event of a fire.

The economy cable assemblies also come in six configurations: down-angle IX A plug to IX A plug, to RJ-45 jack or to RJ-45 plug; and up-angle IX A plug to IX A plug, to RJ-45 jack, or to RJ-45 plug. Protection is provided by a PVC outer jacket rated CMR/CMX.

"Not all cable assemblies with compact connectors are robust but these certainly are," said Product Line Manager Dustin Guttadauro. "You can parallel mount them to fit more connectors in a smaller space, and the entire assembly is well protected for years of industrial use."

L-Com | www.l-com.com

NEW PRODUCT SUBMISSIONS— E-mail: product-editor@circuitcellar.com

PRODUCT NEWS

Penguin Solutions Next-Gen Altus HPC Servers with AMD EPYC 9005 Series Processors

Penguin Solutions, Inc. announces availability of its new Altus servers powered by AMD EPYC 9005 Series processors featuring 192 core count and 12 channels of DDR5 memory tailored to run HPC applications at the highest level of performance. Powered by these latest AMD EPYC processors, Penguin Solutions Altus servers offer best in-class price-to-performance value, and are an ideal match to architect HPC and AI infrastructure, including AI clusters that support mixed-use workloads.

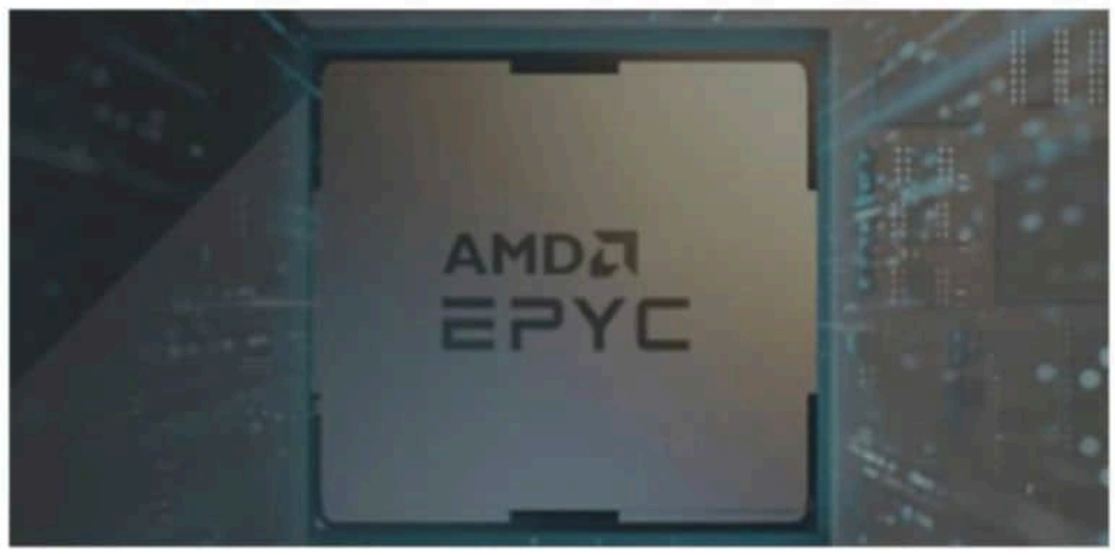
"AMD has delivered processor leadership in the EPYC 9005 Series that drives further HPC and AI performance in meeting our customers' advanced computing needs," said Dara Ambrose, vice president of engineering at Penguin Solutions. "The 5th Generation EPYC processors continue to showcase AMD's innovation in core count and memory architecture, enabling Penguin Solutions to deliver the powerful cluster architectures our customers require to push the envelope in their own simulation, modeling, and discovery designs."

Penguin Solutions has air-cooled models Altus XE1312v2 and Altus XE2312v2 currently available, with liquid-cooled Altus XE2342v2 servers targeted for availability in 1H 2025. Penguin's Altus servers are an option in Penguin's OriginAI infrastructure solution, which combines a full suite of hardware, software, and services that enable enterprises, researchers, and hyperscalers to quickly and reliably deploy and manage HPC and AI compute clusters at scale.

Specifications:

- Altus XE1312v2 (4x PCIe Gen5 x16 slots, up to 6TB (24 DIMMs) DDR5-6400MT/s memory, and up to 400W TDP support)
- Altus XE2312v2 (2x PCIe Gen5 x16 slots, up to 6TB (24 DIMMs) DDR5-6400MT/s memory, and up to 500W TDP support)
- Altus XE2342v2 (8x PCIe Gen5 x16 slots, up to 24TB (96 DIMMs) DDR5-6400MT/s memory, and up to 500W TDP support)

Penguin Solutions | www.penguinsolutions.com



Frontgrade Gaisler Unveils GR716B, a New Standard in Space-Grade Microcontrollers

Frontgrade Gaisler has launched its latest radiation-hardened microcontroller—the GR716B. Building on the success of the GR716A, this latest innovation is a step forward in the development of energy-efficient and adaptable microcontrollers for space applications. As missions grow in complexity and ambition, the industry sees a growing need for technologies that not only deliver dependable performance over long durations, but also have the capabilities to manage a multitude of tasks.

The GR716B is an all-in-one solution for supervision, monitoring, and control in satellite applications. Designed with versatility in mind, the new microcontroller adapts effortlessly to a wide range of space systems due to its comprehensive set of standard interfaces, architectural features, and integrated analogue functions.

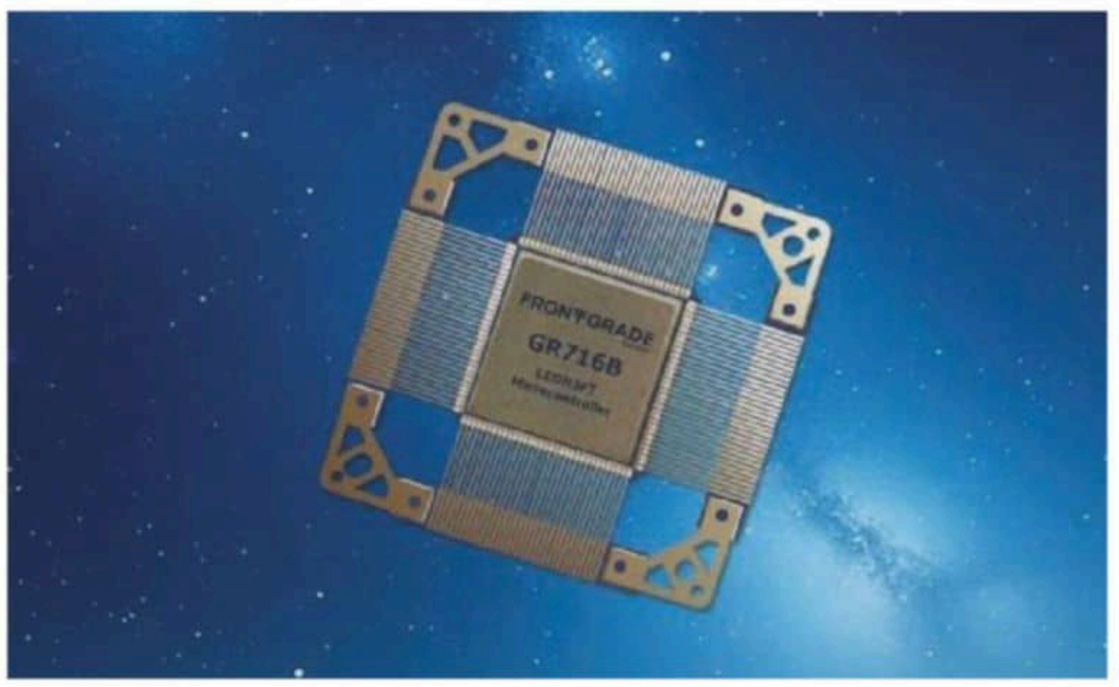
Sandi Habinc, General Manager at Frontgrade Gaisler, remarked, "With the GR716B, we are excited to offer a solution that reflects the growing ambitions of the space industry. The GR716B engineering models are now being made available to alpha customers for integration into new missions."

The GR716B combines proven technology with innovations that address the space industry's most pressing challenges such as the need for low power consumption and efficient use of onboard resources. Frontgrade Gaisler also plans to bring the GR716B System-in-Package (SiP) solution to market, which combines the new microcontroller with non-volatile memory, thereby simplifying system design.

The development of the GR716B was made possible through support from the European Space Agency's (ESA) Core Competitiveness program, part of the Advanced Research in Telecommunications Systems program and within ESA's Connectivity and Secure Communications directorate, and Swedish National Space Agency (SNSA), representing a continued commitment to advancing space-grade technologies.

Domenico Mignolo, ESA Acting Head of Technology and Products Division, said, "We are glad to support Frontgrade Gaisler through our Core Competitiveness programme. By supporting the development of versatile space-grade microcontrollers, ESA aims to bring innovation in satellite communications in order to strengthen industrial competitiveness."

Frontgrade | www.frontgrade.com



NEW PRODUCT SUBMISSIONS— E-mail: product-editor@circuitcellar.com

Saelig Introduces Versatile Rigol 500MHz 16-bit Signal Generator

Saelig Co., Inc. has introduced the Rigol DG5000 Pro Series Signal Generators with up to 2.5GSa/s sampling and 64Mpts/ch memory depth (128Mpts/ch optional). The three models in the DG5000 Pro series are DG5252 Pro (250MHz), DG5352 Pro (350MHz), and DG5502 Pro (500MHz).

The isolated dual-channel DG5000 Pro Series are all-in-one generators that integrate Function Generation, Arbitrary Waveform Generation, Noise Generation, Pulse Generation, Harmonics Generation, and Analog/Digital Modulation. They can be powered from an attached battery pack to provide ground isolation and facilitate field test situations. All the 16-bit vertical resolution Pro models are priced significantly lower than the equivalent 14-bit Rigol DG5000 predecessors and offer improved specifications such as faster sampling and a larger 10.1'' HD touch screen, which allows configuring dual-channel waveforms on a single screen.

Features of the series include:

- Dual-channel isolated outputs
- AM/FM/PM/ASK/FSK/PSK/SUM modulation
- Up to 500MHz signal generation
- Phase alignment
- Adjustable time delay
- 2-channel synchronization
- Pulse width & time-delay adjustable
- Pulse & Square waves >120MHz
- Square: 170MHz max. frequency, 0.8ns min. rise time
- Pulse: 120MHz max. frequency, 4.2ns min. pulse width
- Sequence mode (up to 128 waveforms - complex sequencing)
- Built-in high-order harmonic generator (max. 20th order)
- Max Arb waveform length 64Mpts/ch (128Mpts opt.)
- Software Options: Sequence, IQ modulation, Multi-pulse, Pattern, Multi-tone
- Standard Web Control function for easy remote cooperation
- Supports external modulation

The DG5000 Pro Series Signal Generators offer isolated outputs, superior square wave performance, complex sequence generation, integrated arbitrary waveform creation & editing via a user-friendly UI on a large display. They can simulate test situations supporting optoelectronic device research, logic chip development, power semiconductor development and more.

Saelig Co., Inc. | www.saelig.com

Allegro MicroSystems Unveils Innovative Power Products for a More Energy Efficient Future

Allegro MicroSystems, Inc., a global leader in power and sensing solutions, has announced a groundbreaking series of Power products poised to redefine performance and efficiency across automotive, industrial and data center applications. Allegro's innovative products, debuting at electronica 2024, empower customers to achieve unparalleled performance while simplifying design and reducing costs. The new product lineup not only addresses the escalating demands for higher voltage and power, but also delivers industry-leading efficiency and reliability, marking a significant advance in power electronics technology.

Allegro's comprehensive suite of products encompasses cutting-edge true 48V motor drivers like the A89212, A89224 and A89333 designed to address the thermal management needs of hybrid electric vehicles and AI Servers. Complementing these drivers is the APM81815, a 48V buck regulator designed for superior EMI performance in dual-voltage hybrid electric vehicles. Rounding out the list of new products is the AHV85311, a high-power isolated gate driver designed to accelerate the development of Silicon Carbide (SiC)-based power electronics. From optimizing efficiency in automotive applications to simplifying industrial designs as well as enhancing reliability in data centers, Allegro's new power IC innovations enable engineers to design smarter, more efficient systems.

"The new power IC solutions from Allegro represent a significant leap forward in power electronics design," says Ram Sathappan, Sr. Director of Global Marketing & Applications at Allegro MicroSystems. "We're not just meeting the evolving demands of higher voltage and power; we're redefining what's possible. Our customers demand solutions that simplify design, enhance efficiency and lower costs, all of which our new products deliver. We're excited to partner with our customers to shape the future of power electronics and look forward to showcasing our innovative solutions at electronica."

Allegro's new power and motor control solutions represent a significant step forward in addressing many of the challenges faced in today's rapidly evolving automotive and industrial landscapes.

Allegro MicroSystems | www.allegromicro.com

NEW PRODUCT SUBMISSIONS— E-mail: product-editor@circuitcellar.com

TEST YOUR EQ

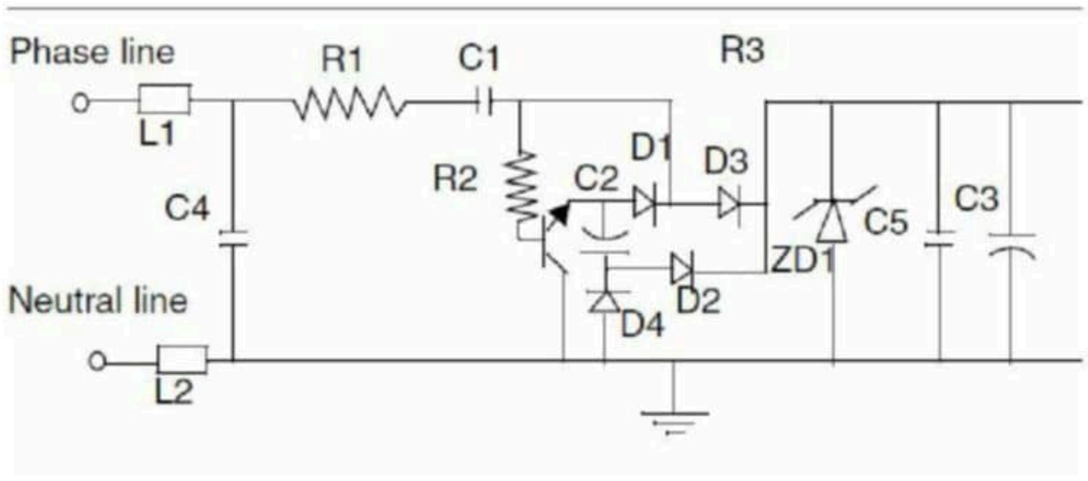
Contributed by David Tweed

Question 1 —

You are designing a power supply for a battery-operated device. You have a choice of two batteries with the same energy capacity. One has a higher voltage than the target output voltage, requiring a buck converter, and the other has a lower voltage than the target, requiring a boost converter. Is there a reason to choose one over the other?

Question 2 —

This rather confusingly-drawn schematic represents a non-isolated DC power supply connected directly to the AC mains. What exactly are the transistor and C2 doing in this circuit?



Question 3 —

Why wasn't an ordinary 4-diode bridge rectifier used instead?

Question 4 —

An application requires a circuit that generates N pulses each time it is triggered. The pulses need to be at a 10 Hz rate, 50% duty cycle. N varies from 2 to 9, and is fixed for any given instance of the circuit — it does not to be variable.

The modern answer would be to use a tiny 6- or 8-pin MCU. But suppose that external constraints require using only discrete transistors, diodes, resistors and capacitors. Can you come up with a circuit that meets the requirements? How many transistors does it require?

TEST YOUR EQ

qSmartAI80_CUQ610

CELEBRATING 20 YEARS OF CAMERA ENGINEERING

e-con Systems™
Think Camera. Think e-con.

Fast Track your Edge AI Development

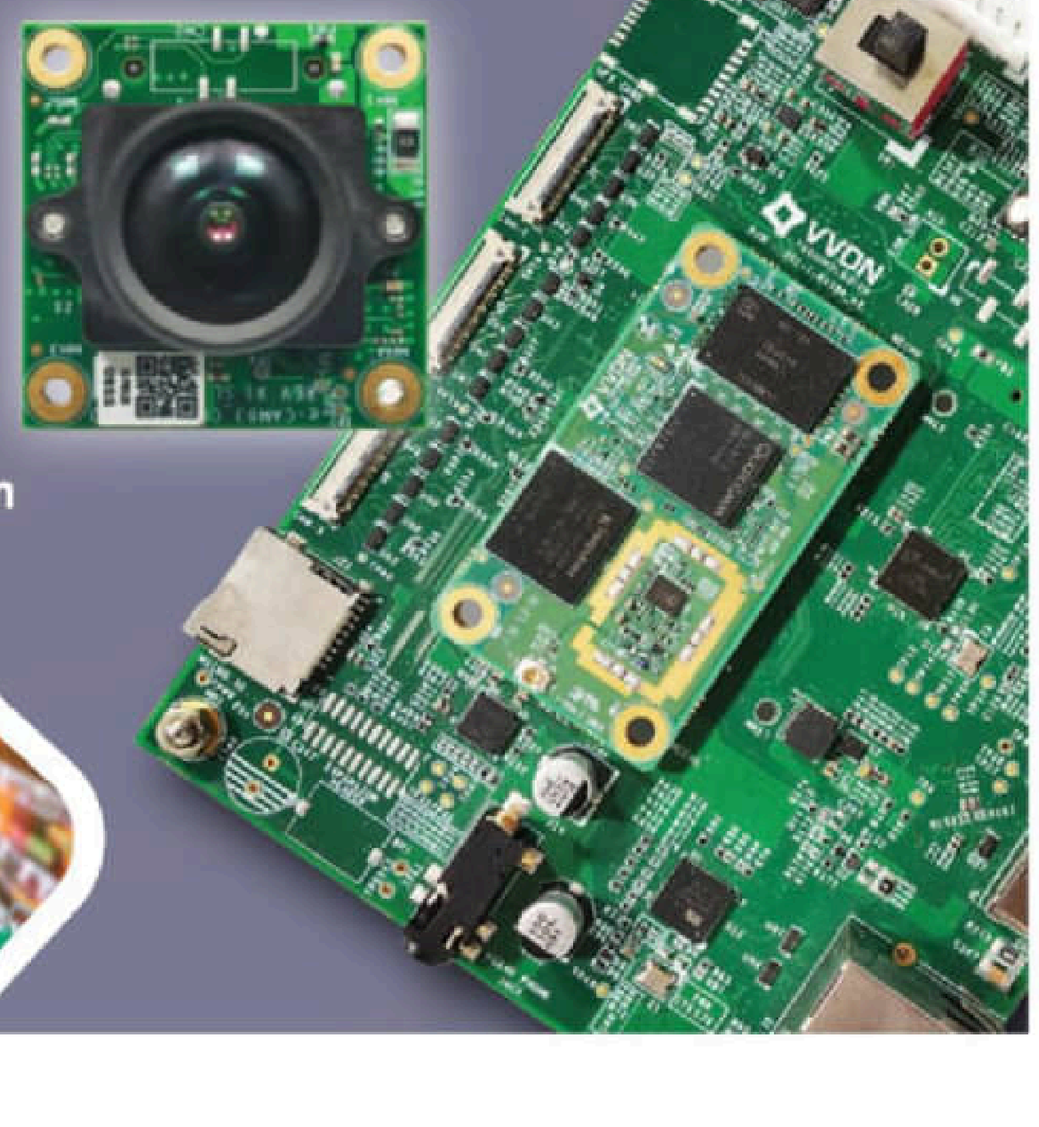
with Qualcomm QCS610 based vision kit

Finetuned Qualcomm ISP

Sony STARVIS IMX415

Neural Processing SDK

Complete Customization



IDEA BOX

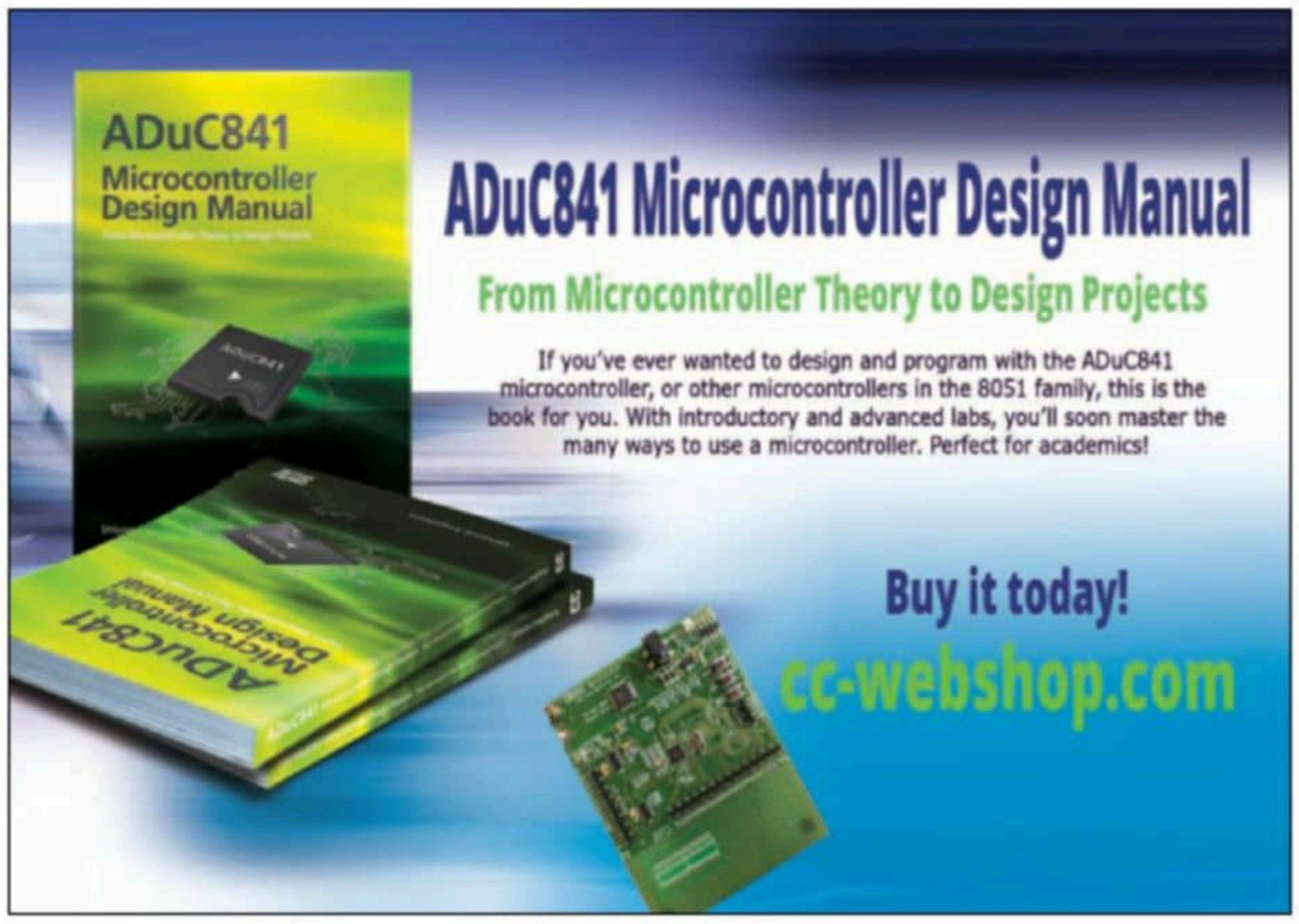
The Directory of PRODUCTS & SERVICES

AD FORMAT:

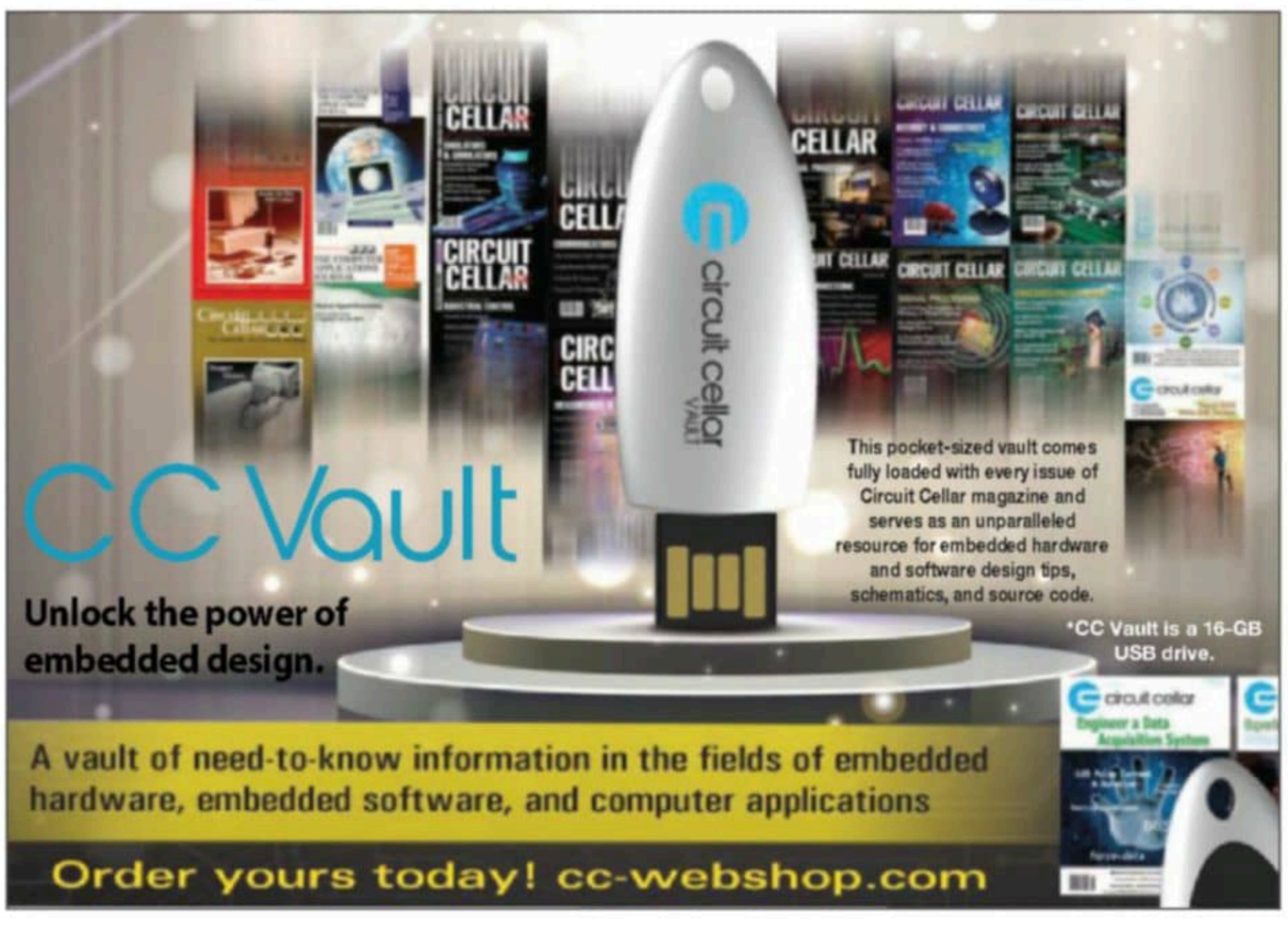
Advertisers must furnish digital files that meet our specifications (www.circuitcellar.com/mediakit).

All text and other elements MUST fit within a 2" x 3" format.
E-mail adcopy@circuitcellar.com with your file.

For current rates, deadlines, and more information contact Hugh Heinsohn at 757-525-3677 or Hugh@circuitcellar.com


ADuC841
Microcontroller Design Manual
ADuC841 Microcontroller Design Manual
From Microcontroller Theory to Design Projects
If you've ever wanted to design and program with the ADuC841 microcontroller, or other microcontrollers in the 8051 family, this is the book for you. With introductory and advanced labs, you'll soon master the many ways to use a microcontroller. Perfect for academics!
Buy it today!
cc-webshop.com

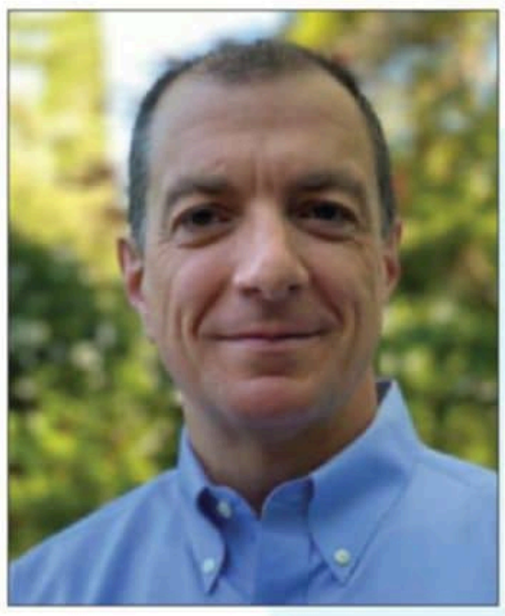
RAPID18
Universal Development Kit for PIC18 Family
ONLY $99!
• Real-time Clock/Calendar with Supercap
• USB port for text communication to the running program
• Built-in bootloader for loading code
• Excellent for AC/DC Data Logging
Kit includes everything you need to develop with a PIC® MCU:
• Rapid 18 Prototyping Board
• Single-Chip IDE C Compiler
• Exercise Book
• Power Adapters and Cables
sales@ccsinfo.com (262) 522-6500 x35
www.ccsinfo.com/cc1223


CC Vault
Unlock the power of embedded design.
This pocket-sized vault comes fully loaded with every issue of Circuit Cellar magazine and serves as an unparalleled resource for embedded hardware and software design tips, schematics, and source code.
*CC Vault is a 16-GB USB drive.
A vault of need-to-know information in the fields of embedded hardware, embedded software, and computer applications
Order yours today! cc-webshop.com

The Future of IoT Sustainability

Atmosic Solving Key IoT Challenges

Cutting Battery Usage by Advancing Sustainability and Interoperability



Battery-powered IoT solutions have a side effect that creates harmful waste and presents barriers to sustainability. By integrating SoCs with energy-efficient IP to reduce power consumption, adding energy harvesting capabilities, and incorporating Matter for interoperability, the push for a battery-free future is taking shape.

By
Paul Davis
Vice President of Product Management and Marketing
Atmosic Technologies

Connected devices are everywhere, in our homes, vehicles, workplaces, factories, and beyond. By 2030, the IoT is set to reach over 40 billion connected devices globally [1] growing at an annual rate of about 20%. However, even with this impressive growth, there are still challenges that could hold back the IoT from reaching its full potential: battery usage, sustainability, and interoperability. This article will explore how Atmosic is helping to address these challenges to protect the planet and reduce maintenance time and costs.

BATTERY USAGE AND SUSTAINABILITY

Many of the IoT solutions that will be deployed between now and 2030 will be battery powered, either in locations where power is not available or attached to mobile devices. Reducing battery waste needs to be a priority for every company designing connected devices since billions of batteries are being thrown away each year.

Batteries can take over 100 years to decompose and leach harmful chemicals into the soil and ground water. Lithium batteries can even cause fires when they are not disposed of correctly. This amount of environmental waste will only continue to grow if the tech industry does not reduce the need for battery replacement in connected devices.

Beyond the desire to go green, there are other advantages in reducing battery use. When businesses have large deployments of IoT devices, there is an operational impact when a battery dies and a sensor or tracking device goes offline. In some cases, the expense of replacing a battery exceeds the original cost of the device itself. For consumers, it's a hassle to replace the batteries in all the various devices we've come to rely on every day. It's annoying to find the remote control not working when you sit down to watch a movie. The problem only gets worse as more battery powered devices like door locks, sensors, and controllers find their way into the connected home.

It is imperative that commercial and consumer products are designed to maximize battery life. Companies are deploying Atmosic's technology to reduce or eliminate the need for batteries in connected devices, minimizing their impact on the environment.

FIGURE 1
Atmosic's ATM34/e high-performance system-on-chip (SoC) series with multiprotocol support, designed for ultra-low power end IoT devices. (Image Source: Atmosic)

DESIGNING FOR ENERGY EFFICIENCY

Designing energy efficient devices starts with using a system on chip (SoC) solution with very low power consumption. Atmosic's

latest ATM34/e [2] series of SoCs in **Figure 1**, uses at least 50% less energy than competitor solutions in many typical IoT applications. There are additional features available that can further reduce power consumption to increase the power savings to a factor of 5 or even 10 when compared to traditional solutions.

To start, Atmosic has a patented design approach to reduce the power consumption of the radio and analog parts of the device. This technology minimizes the energy required for both the RF transmit and receive functions, leading to active power consumption three to four times lower than the competition.

In addition to the unique design of the radio, there are two other functional features, shown in **Figure 2**, that can be leveraged to further reduce the power consumption of IoT devices.

First is the on-demand wakeup receiver (WURX) integrated into the device. This ultra-low power receiver allows the rest of the device to stay asleep while listening for a specific RF transmission signature. This signature is programmable so that individual devices can be woken up on-demand or as part of a group of devices triggered together. For applications where devices are mostly idle and can be triggered to report or act when triggered, this feature can offer significant power savings.

For sensor applications, Atmosic has another power reducing feature called SensorHub. SensorHub is a separate autonomous part of the SoC that monitors a sensor interface without assistance from the CPU, keeping power consumption to a minimum. The SensorHub can be programmed to store sensor data or trigger a wakeup when a sensor reading falls outside of a prescribed range. During normal operation, the sensor will stay in the lowest power state possible and only report on an exception basis.

Beyond helping to extend battery life, Atmosic's wakeup receiver and SensorHub have additional benefits in IoT applications. In environments with many devices actively operating, decreasing the amount of RF traffic by going to an on-demand or exception-based reporting scheme improves system performance by reducing the probability of RF interference. In addition, the system-wide loading of the network and cloud services is reduced, which can also improve the operational costs as well.

ENERGY HARVESTING POWER

To move to a completely battery free and maintenance free model for IoT devices, developers can consider leveraging energy harvesting technology. Most people think of large solar panels powering houses when they hear the term energy harvesting, but a better model for the IoT are the battery-free desktop calculators powered by photovoltaic (PV) cells harvesting indoor light.

Adding energy harvesting technology requires consideration of the available energy source, which could be ambient light, mechanical/kinetic motion, thermal or even radio frequency (RF) energy. The ATM34e SoC has an integrated power management unit (PMU) capable of capturing energy from harvesting elements in addition to managing batteries and rechargeable storage devices. This integrated PMU is both more power-efficient and cost-efficient when compared to traditional solutions that leverage an independent energy harvesting PMU.

This solution, shown in **Figure 3**, offers great flexibility when combining harvesting and storage methods. For instance, one can use harvested energy to extend the life of a standard battery, powering the device when harvested energy is available, storing excess harvested energy in a supercapacitor, and only using the battery when both harvested and stored excess energy is not available. Another option is having harvested energy trickle charge a rechargeable battery, ensuring that the device will remain operational even when harvested energy is not available for long periods of time. With this approach it is possible to never replace a battery for the lifetime of the product.

Finally, for some implementations, it is possible to operate entirely without any batteries. If harvested energy is available on a consistent basis or if the operational model requires operation when harvested energy is available, batteries can be eliminated from the product design.

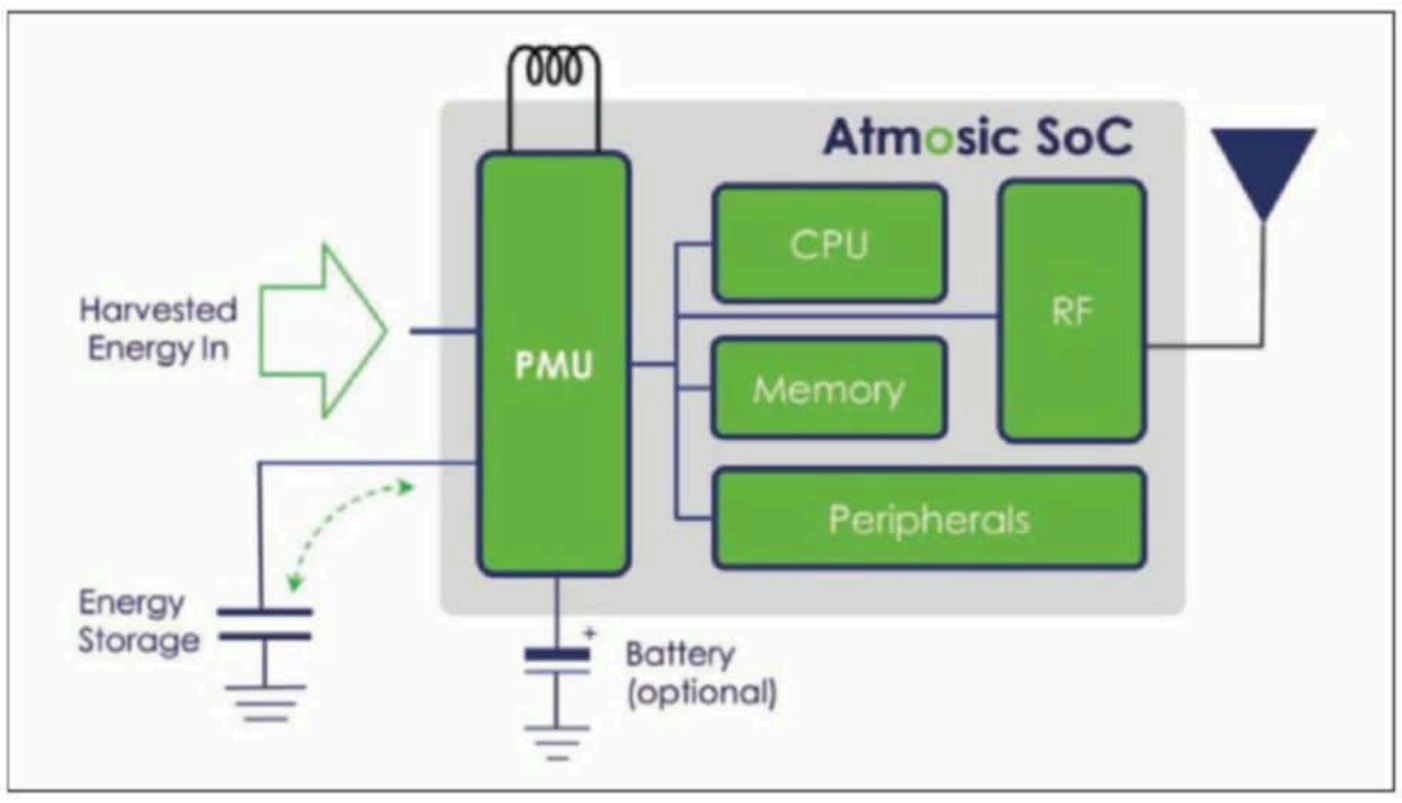


FIGURE 2
ATM34 Integrated Energy Harvesting and Storage Solution. A single SoC incorporates an energy harvesting power management unit that also manages multiple energy storage devices with CPU, radio, and peripheral interface functions. (Image source: Atmosic)

IMPROVING IOT INTEROPERABILITY

As we see more connected devices in our homes, cities and workplaces, effective communication between devices is crucial for a good user experience. When devices don't interoperate, the consumer is left with a bad experience, even feeling locked into a specific set of devices or platform. The good news is that the IoT industry is embracing standards like Matter to address this problem.

The Matter standard allows smart home devices to seamlessly communicate, regardless of brand. Think of a smart home hub that will communicate with smart thermostats, light bulbs, windows shades, and other sensors; if these devices can talk to each other, you can talk to your hub to control the temperature or lighting in the family room as you settle in for movie night. The same technology can be added to the remote control on the coffee table. Now you can easily control your TV, lights, blinds, security system, and other devices. Atmosic's ATM34/e series has built in support for Matter to make this scenario a reality.

By making devices sustainable, efficient, and interoperable, we'll see a new era of IoT innovation that benefits consumers, businesses, and the environment.

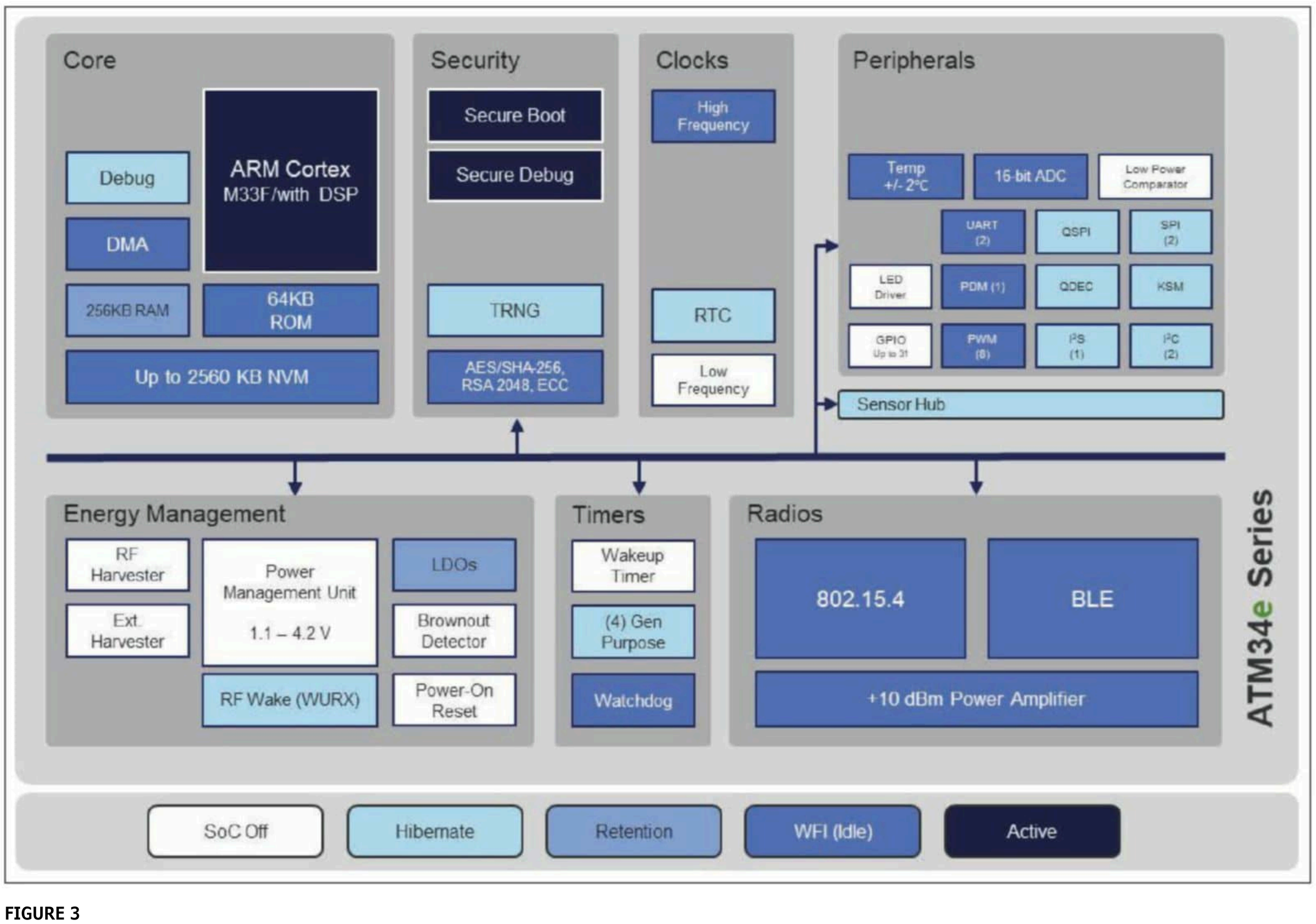


FIGURE 3
This ATM34 block diagram depicts all the parts of the SoC: The CPU, memory, security, clocks and timers, peripheral, power management, and radio sections are all depicted with indications of the minimum power state for all devices to be active. (Image Source: Atmosic)

Additional materials from the author are available at: www.circuitcellar.com/article-materials References [1] and [2] as marked in the article can be found there.

Resources

Atmosic Technologies | www.atmosic.com

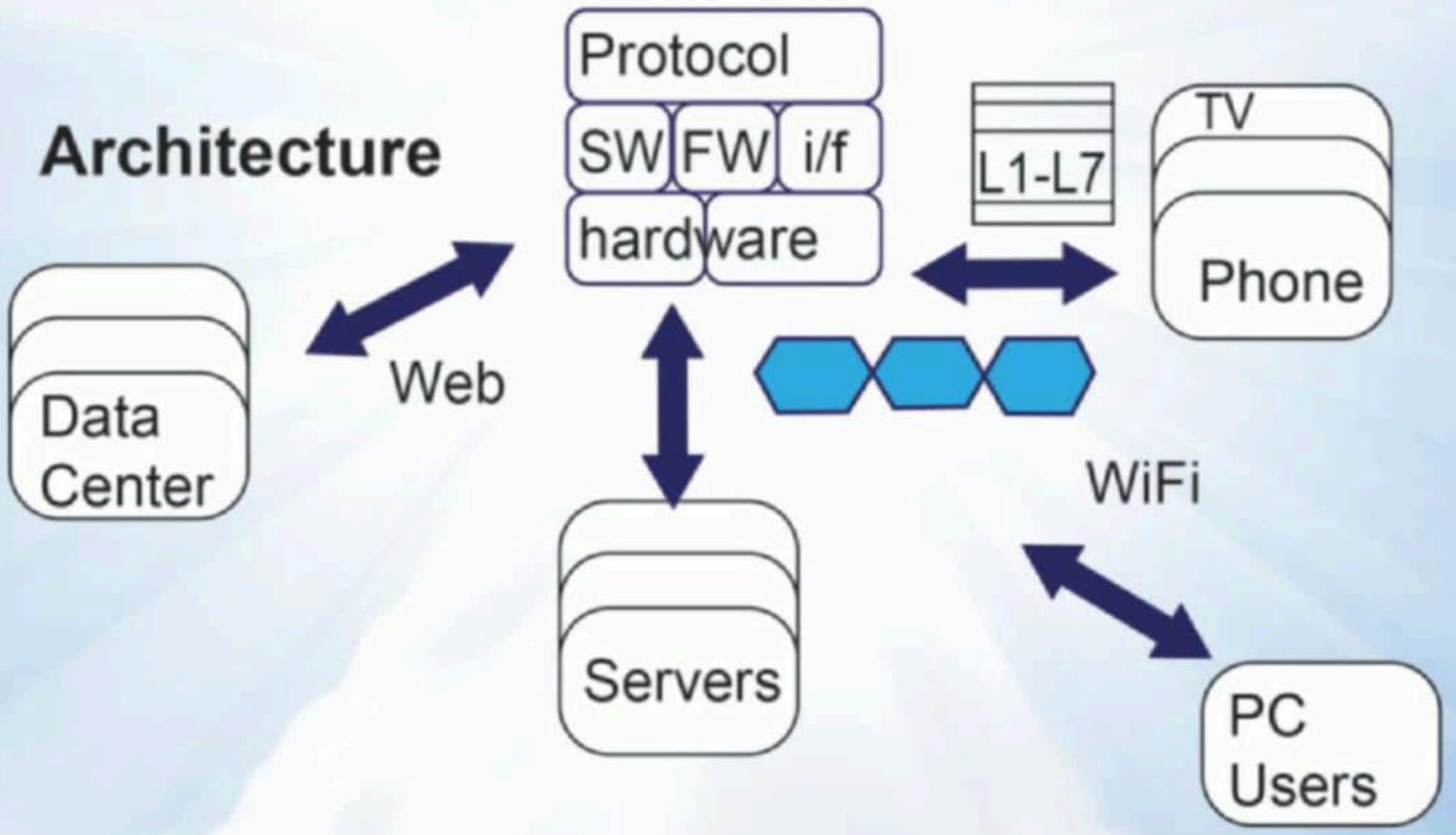
ABOUT THE AUTHOR

Paul Davis is vice president of product management and marketing at Atmosic Technologies. For over 25 years he has been involved in the definition, development, and delivery of wireless technologies for a wide range of industries. His current focus at Atmosic is the integration of energy harvesting and low power wireless semiconductors for various IoT applications. Paul previously worked in a variety of senior product management, marketing and engineering roles at both large and small companies in Silicon Valley. Paul has a BS in Physics from MIT and a MS in Electrical Engineering from University of California, Santa Barbara.

19842860

SeaFire® Micros, Inc.

IP & Technology for Cloud Networks



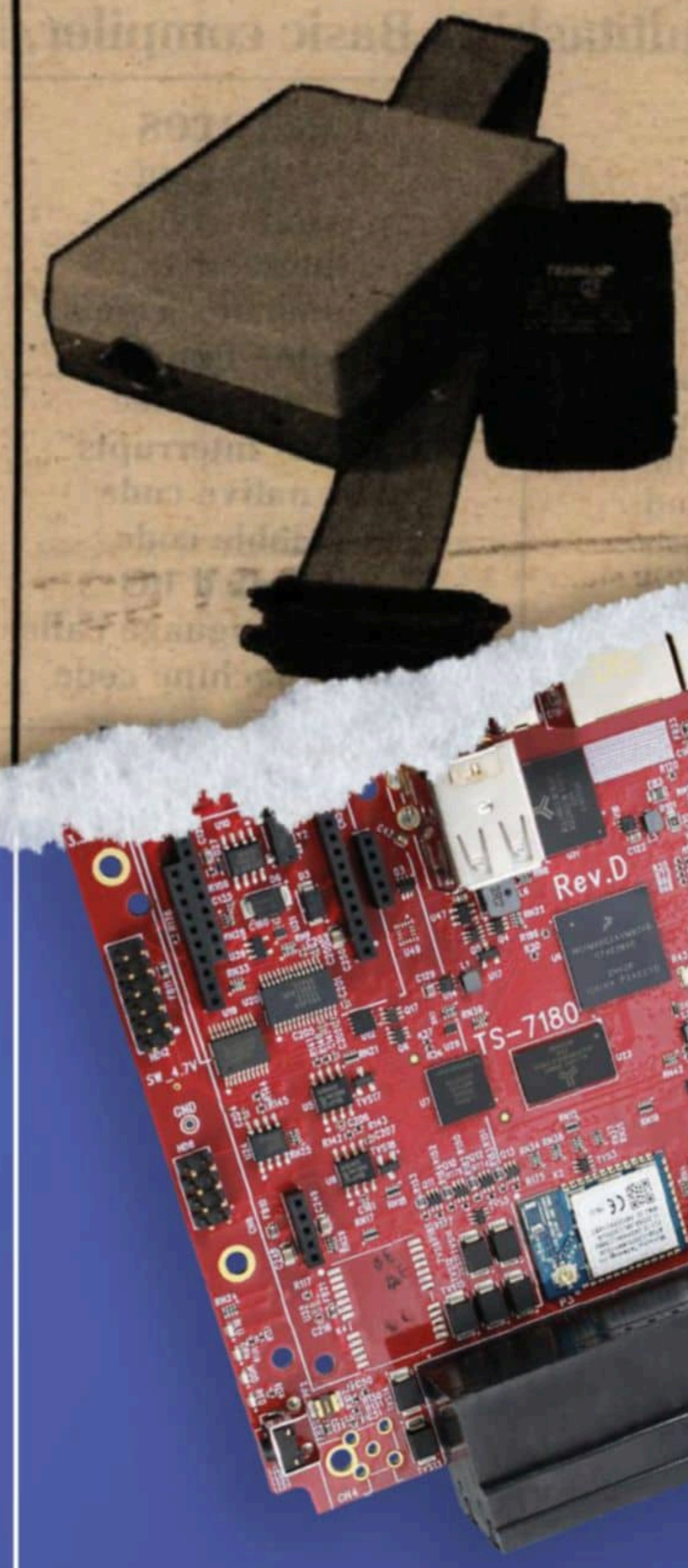
Design through Production
Military — Commercial

Hardware (Analog/Digital)
Firmware • Software • FPGA

Patented Network Technology | Unique Cloud Technology
No Compete Government Procurement
No Compete Government Funding

20% DISCOUNT on 1st Purchase

info@seafire.com | seafire.com



Technologic Systems

SERIAL to PARALLEL

Apple //c, DEC RAINB

Epson, IB

and other serial inte

Start

Printer B

TS-7180

This versatile embedded single board computer hits on all the main points for a low power, cost effective, Internet-of-Things (IoT) capable, and ready-to-deploy OEM board with an emphasis on data integrity.

NXP i.MX6UL SOC with a 696 MHz Arm® Cortex®-A7 core

512 MB DDR3 RAM (1 GB DDR3 RAM Option Available)

4 GB MLC eMMC Flash, configurable as pSLC for added reliability

Connected with WiFi, Bluetooth, MultiTech and XBee/NimbeLink

Easily Track Assets with GPS Radio and Cellular Modem

Celebrating 40 Years

From humble beginnings in a garage in Ohio, EmbeddedTS has grown to become a leader in rugged embedded computing solutions. Since 1984, we've proudly remained family-owned and dedicated to delivering high-quality, reliable products.

Every board we sell is still designed, shipped, and supported from right here in the United States, ensuring you get top-tier craftsmanship.

Made in the USA
with global parts

embeddedTS

embeddedTS.com